Sistemas Informáticos
Curso 2006 - 2007

Implementación HW de un gestor de ejecución para sistemas dinámicamente reconfigurables

Juan Antonio Clemente Barreira
José Luis García Casado
Carlos González Calvo

Dirigido por:
Jesús Javier Resano Ezcaray
Dpto: Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

ÍNDICE DE IMÁGENES Y TABLAS

RESUMEN DEL PROYECTO Y PALABRAS CLAVE

RESUMEN DEL PROYECTO

El HW reconfigurable se puede utilizar para construir un sistema multitarea en el que las tareas puedan asignarse en tiempo de ejecución a los recursos reconfigurables según las necesidades de las aplicaciones. En estos sistemas, las tareas se representan normalmente como grafos de subtareas, donde una subtarea es la unidad de planificación. Normalmente, un procesador empotrado controla la ejecución de este tipo de sistemas trabajando con estructuras de datos complejas, como grafos o listas enlazadas, cuyo manejo a menudo genera retardos en la ejecución. Además, las comunicaciones HW/SW son a menudo un cuello de botella del sistema. Por tanto resulta muy interesante reducir tanto los cálculos que realiza el procesador como las comunicaciones. Para lograr este objetivo hemos desarrollado un gestor HW que controla la ejecución de grafos de subtareas en un conjunto de unidades reconfigurables. Este gestor recibe como entrada los grafos junto con una planificación asociada a cada subtarea y garantiza su correcta ejecución sin necesidad de ninguna otra intervención por parte del procesador. Además, incluye mecanismos para optimizar la gestión de las reconfiguraciones reduciendo las penalizaciones que generan en tiempo de ejecución.

ABSTRACT

Reconfigurable HW can be used to build a hardware multitasking system where tasks can be assigned to the reconfigurable HW at run-time according to the requirements of the running applications. Normally the execution in this kind of systems is controlled by an embedded processor. In these systems tasks are frequently represented as subtask graphs, where a subtask is the basic scheduling unit that can be assigned to a reconfigurable HW. In order to control the execution of these tasks, the processor must manage at run-time complex data structures, like graphs or linked list, which may generate significant execution-time penalties. In addition, HW/SW communications are frequently a system bottleneck. Hence, it is very interesting to find a way to reduce the run-time SW computations and the HW/SW communications. To this end we have developed a HW execution manager that controls the execution of subtask graphs over a set of reconfigurable units. This manager receives as input a subtask graph coupled to a subtask schedule, and guarantees its proper execution. In addition it includes support to reduce the execution-time overhead due to reconfigurations. With this HW support the execution of task graphs can be managed efficiently generating only very small run-time penalties.

PALABRAS CLAVE

Hardware multitarea, reconfiguración en tiempo de ejecución, gestor de tareas, FPGA, ISE, EDK, Virtex II-PRO.

AUTORIZACIÓN

Los ponentes Juan Antonio Clemente Barreira, con DNI 52887871, José Luis García Casado, con DNI 51462262, y Carlos González Calvo, con DNI 77811220, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos tanto la propia memoria, como el código, la documentación y el prototipo desarrollado.

Juan Antonio Clemente Barreira

José Luis García Casado

Carlos González Calvo

En Madrid, a 2 de julio de 2007

INTRODUCCIÓN

El proyecto de la asignatura Sistemas Informáticos denominado "*Implementación HW de un gestor de ejecución para sistemas dinámicamente reconfigurables*" consiste en la realización de un gestor para la correcta ejecución de una tarea, representada en forma de grafo de subtareas, con respecto a una planificación dada.

CONTRIBUCIÓN DEL PROYECTO

La principal contribución de esta memoria es la descripción de un sistema para la gestión eficiente de HW reconfigurable. Este gestor controla la ejecución de una serie de subtareas en un conjunto de unidades reconfigurables siguiendo un grafo de subtareas dado y una planificación. Esta información se almacena en una tabla asociativa y un conjunto de FIFOs respectivamente.

Además, ya que la latencia de reconfiguración puede generar retardos significativos en la ejecución (para las FPGAs estos retardos son del orden de milisegundos [7]), el gestor aplica dos técnicas para reducir estas penalizaciones. En primer lugar, antes de reconfigurar una UR para cargar una subtarea en ella, el gestor comprueba si esa subtarea había sido cargada allí previamente. Si esto ocurriese, la configuración puede ser reutilizada directamente sin necesidad de cargarla. En segundo lugar, el gestor aplica una técnica de anticipación [8] para ocultar las latencias de reconfiguración en la medida de lo posible.

Dado que, planificar las reconfiguraciones es algo crítico para un sistema HW multitarea, y muchos autores han propuesto algunas heurísticas específicas que consiguen resultados casi óptimos [9]. Para proporcionar soporte a estos planificadores, el gestor se puede programar para que siga una secuencia de reconfiguraciones dada (aproximación *basada en una planificación*). En este caso, el gestor identifica cuál es la tarea que debe reconfigurarse a continuación de acuerdo a la planificación dada (que está almacenada en una FIFO), y comienza su reconfiguración tan pronto como sea posible.

OBJETIVOS Y LOGROS REALIZADOS

Los objetivos iniciales del proyecto incluían el diseño del gestor, su implementación en VHDL o verilog, su ejecución sobre una placa de FPGA y la realización de una comparativa con un gestor SW equivalente.

Tras tener claro una visión global del sistema, se repartió el gestor en diferentes unidades quedando fijadas las tareas que debía realizar cada una y las interacciones entre ellas. En el proceso de desarrollo del proyecto, se simultaneó el diseño de una nueva unidad con la implementación y testeo en ISE de la unidad anteriormente diseñada, lo que permitió tener una primera versión en relativamente poco tiempo. El lenguaje escogido para la implementación del gestor fue VHDL ya que, era el lenguaje más conocido por los miembros del grupo.

Para comprobar la correcta funcionalidad de los diseños realizados, el área qe ocupan y su rendimiento, todos los módulos fueron sintetizados y comprobados desarrollando bancos de pruebas para las herramientas de simulación Post Place & Route en el Ise 9.1i. Estas simulaciones garantizan el mismo resultado que una ejecución en la FPGA final, pero resulta mucho más cómodo trabajar con ellas al poder definirse de forma sencilla todo tipo de bancos de pruebas.

Finalmente, se elaboró el código en C equivalente al gestor y se ejecutó en un POWERPC y en MicroBlaze con ayuda de la herramienta EDK 9.1. Los resultados obtenidos nos confirmaron la rapidez del sistema HW frente al SW.

Adicionalmente, la realización de este proyecto ha dado lugar a la publicación de dos artículos:

- **"HW implementation of an execution manager for reconfigurable systems"**, Javier Resano, Juan Antonio Clemente, Carlos Gonzalez, Jose Luis Garcia and Daniel Mozos, Engineering of Reconfigurable Systems and Algorithms (ERSA), 2007.

- **"Un sistema para la gestión eficiente del HW reconfigurable",** Carlos González, Juan Antonio Clemente, José Luis García, Javier Resano y Daniel Mozos, Jornadas sobre Computación Reconfigurable y Aplicaciones (JCRA), 2007.

ESTRUCTURA DEL DOCUMENTO

El presente documento está dividido en dos partes bien diferenciadas. En la primera se realiza una aproximación tecnológica de todos los dispositivos, técnicas de reconfiguración y software involucrados en la realización del proyecto. Se describen en detalle las características y funcionalidades de la FPGA, y las herramientas de diseño empleadas para la realización del proyecto: *Xilinx ISE 9.1i, ModelSim 6.0a* y *Xilinx EDK 9.1i*. Finalmente, se ofrece una visión general del HW reconfigurable, sus alternativas de diseño y cómo y por qué es interesante utilizalo para implementar un sistema con varias unidades reconfigurables.

En la segunda parte de la memoria, se describe todo el diseño del proyecto. Se explican todas las unidades en que se puede dividir el gestor: describimos las FIFOs utilizadas, la tabla de tareas (entrada de la tabla, red iterativa, HW de control y finalmente, su esquemático a alto nivel), los módulos que caracterizan a las unidades reconfigurables, las FIFOs de reconfiguraciones y de eventos, el árbitro que controla las escrituras en la FIFO de eventos y la unidad de control. Asimismo, para comprender mejor el funcionamiento del sistema, explicamos la ejecución de un grafo de subtareas paso a paso. Por otro lado, se realiza un análisis comparativo para conocer las ventajas que proporciona este diseño hardware sobre un control meramente software. Finalmente, exponemos algunas conclusiones y posibles ampliaciones del proyecto que se pueden realizar como trabajo futuro.

Como apéndice, se muestra el código VHDL desarrollado, el código C utilizado para la versión software comparativa y un glosario de términos.

APROXIMACIÓN TECNOLÓGICA

FPGA

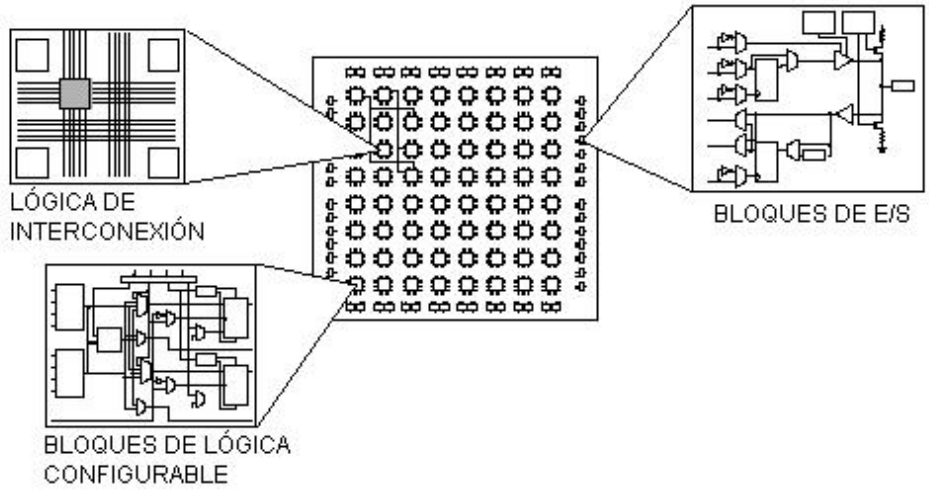
Generalidades

FPGA es el acrónimo de *Field-Programmable Gate Array*. Es un dispositivo hardware dinámicamente reconfigurable, donde se pueden programar infinidad de diseños digitales distintos.

Las FPGAs son una alternativa superior a las mask-programmed ASCIs (Application-Specific Integrated Circuit), con las cuales es imposible hacer actualizaciones de diseño sin reemplazamiento de hardware.

Tres son las principales partes de toda FPGA: los Bloques de Lógica Configurables (Configurables Logic Blocks, CLBs), los Bloques de Entrada/Salida (Input/Output Blocks, IOBs) y toda la lógica de interconexión de los elementos funcionales de la FPGA (figura 1). A continuación se tratará brevemente de cada parte.

Figura 1. Bloques funcionales principales de la FPGA



Cada CLB tiene un conjunto de celdas lógicas (Logic Cells, LC). Un LC contiene un generador de funciones implementado, a su vez, con un look-up table que puede programarse para realizar distintas funciones. Además de los cuatro LCs, cada CLB contiene lógica que combina los generadores de funciones para permitir funciones de mayor complejidad.

Los IOBs proporcionan interfaz entre los pines externos y la lógica interna. Cada IOB controla un pin, que puede ser configurado como entrada, como salida o como pin bidireccional. Además cada uno tiene tres registros que comparten la señal de reloj (CLK), pero con distintas señales de capacitación de reloj (CE).

Con la lógica de interconexión concluye la descripción de los elementos funcionales. Su principal característica es que las interconexiones están dirigidas, como el resto de los elementos lógicos, por los valores guardados en las celdas de memoria internas.

Una FPGA debe ser configurada para poder simular un diseño digital, donde configuración se define como el proceso mediante el cual el bitstream (flujo de unos y ceros) de un diseño se carga en la memoria de configuración de la FPGA programando tanto los CLBs como las

interconexiones y la E/S. La parte central del proyecto consistió en realizar esta labor, en programar o configurar una FPGA mediante el envío de señales generadas por software.

La FPGA con la que se ha trabajado es la XC2VP30-5 FG676C de familia Virtex-II PRO de Xilinx. La familia Virtex-II se caracteriza por cargar los datos de configuración en celdas internas de memoria estática. Los valores guardados en estas celdas determinan las funciones lógicas y las interconexiones implementadas en la FPGA. El hecho de guardar los valores dentro de estas celdas permite infinitas reprogramaciones del dispositivo. Además en esta FPGA la configuración puede cambiarse en tiempo de ejecución (reconfigurción dinámica) lo que posibilita la realización de un sistema multiprocesador hardware con varias unidades reconfigurables y uno o varios procesadores.

El nombre de la FPGA, XC2VP30-5 FG676C, indica lo siguiente:

- **Tipo de dispositivo:** XC2VP30, dispositivo 30 de la familia Virtex-II PRO.
- **Grado de velocidad:** 5, estándar.
- **Tipo de embalaje:** FG, Fine Pitch BGA 27 x 27 mm, 1.0 mm ball pitch.
- **Número de pines:** 676.
- **Rango de temperatura:** C, comercial (entre 0º y 85º).

Dentro de las distintas arquitecturas dinámicamente reconfigurables actuales, las FPGAs dominan ampliamente el mercado. La razón principal es que se basan en una tecnología madura con muchos años de desarrollo y sustentada por un amplio conjunto de herramientas de diseño. Por esta razón, el trabajo realizado en este proyecto ha tomado a las FPGAs como punto de referencia a la hora de desarrollar los prototipos y evaluar los resultados.

Xilinx Virtex-II Pro

Xilinx es una empresa especializada en el desarrollo de dispositivos programables (FPGAs, CPLDs, etc.) desde 1985. El modelo de FPGA utilizado en este proyecto es el Virtex-II Pro, desarrollado por Xilinx en 2004. A continuación se enumeran algunas de sus características principales:

- Arquitectura lógica programable
  - Tecnología de 130nm y 9 capas de cobre
  - Desde 3000 hasta 99000 celdas lógicas
  - Velocidad de reloj superior a los 400MHz
  - Alto rendimiento y bajo consumo
- Escalabilidad. Hasta 11 formatos.
- Características avanzadas
  - Memoria distribuida y empotrada (Flash)
  - Gestión de relojes digitales
  - Reconfiguración de la FPGA total/parcial en función de la actualización de los productos del mercado
  - Tecnología XCITE que permite mejorar la integridad de la señal y reducir espacio
- Conectividad
  - Hasta 20 *serial transceivers* (RocketIO) *full-duplex* (de 622Mbps hasta 3.125Gbps)
  - Conexiones a 100MHz mediante LVDS
- Procesamiento avanzado
  - Dos procesadores empotrados IBM PowerPC 405 a 400MHz
- Herramientas de desarrollo
  - Herramientas para la programación de la FPGA

VHDL

El VHDL (*Very High Speed Integrate Circuit Hardware Description Language*), es un lenguaje de descripción y modelado, diseñado para describir la funcionalidad y la organización de sistemas *hardware* digitales, placas de circuitos, y componentes.

La finalidad del modelado es la simulación. La sintaxis amplia y flexible del lenguaje VHDL permite tanto el modelado estructural como el modelado funcional de circuitos. En el primer caso, se describe el circuito indicando los componentes y las conexiones que lo componen (lo cual requiere un conocimiento detallado del circuito). En el segundo caso, se describe el circuito indicando lo que hace y cómo funciona, es decir, describiendo su comportamiento (sin necesidad de conocer su estructura interna).

Esta segunda metodología de modelado resulta muy interesente desde el punto de vista del diseño de sistemas digitales. Más aún teniendo en cuenta que hoy en día otra de las aplicaciones del lenguaje VHDL, con una gran demanda de uso, es la síntesis automática de circuitos. En el proceso de síntesis, se parte de una especificación de entrada con un determinado nivel de abstracción, y se desciende verticalmente por los niveles de la jerarquía de diseño hasta llegar a una implementación más detallada, menos abstracta. Puesto que el VHDL fue inicialmente concebido para el modelado de sistemas digitales, su utilización en síntesis no es inmediata. Sin embargo, la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

Así pues, VHDL permite diseñar, modelar y comprobar un sistema desde un alto nivel de abstracción hasta el nivel de definición estructural de puertas lógicas. Además, siguiendo ciertas guías para síntesis, permite la implementación de diseños a nivel de puertas lógicas. Al estar basado en un estándar (IEEE Std. 1076-1987) reduce errores de comunicación y problemas de compatibilidad. Finalmente, dada su característica de modularidad, permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.
Los componentes de un proyecto VHDL son los siguientes:

- **Entity**: Es el más básico de los bloques de construcción en un diseño. Una entidad VHDL especifica el nombre de la entidad, sus puertos, e información relacionada con ella. Todos los diseños son creados usando una o varias entidades. La entidad describe la interfaz en el modelo VHDL.

- **Architecture**: La arquitectura describe la funcionalidad esencial de la entidad y contiene los estados que modelan el funcionamiento de ésta. Una entidad puede tener varias arquitecturas.

- **Configuration**: Permite unir la instancia de un componente a la pareja entidad-arquitectura. Describe el comportamiento a utilizar para cada entidad.

- **Package**: Es una colección de los tipos de datos y subprogramas usados comúnmente en un diseño. Las librerías forman parte de los *packages*.

PAQUETES SOFTWARE UTILIZADOS

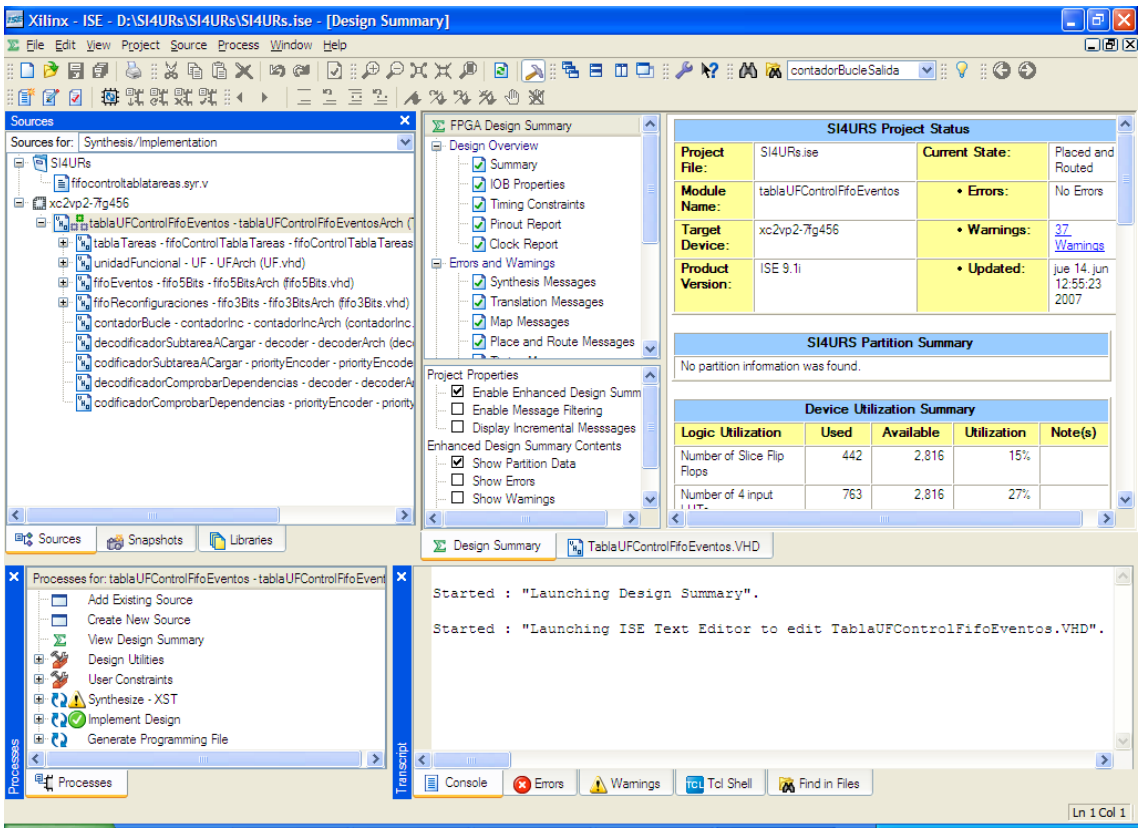
Para el desarrollo de la implementación en VHDL del gestor y su posterior testeo, se utilizaron los programas Xilinx ISE 9.1i y ModelSim 6.0a respectivamente. Posteriormente, se utilizó el programa Xilinx EDK 9.1i para realizar un SW equivalente sobre PowerPC y Microblaze.

Xilinx ISE 9.1i

El entorno de desarrollo ISE de Xilinx posee un aspecto similar al de los entornos de programación actuales como puede ser Visual Basic o Visual C, es decir, posee diversas

ventanas para la visualización de tareas específicas sobre cada una de ellas. En este caso existen cuatro tipos de ventanas (figura 2):

Figura 2. Pantalla principal del entorno Xilinx ISE



1. Ventana de ficheros fuente. En esta ventana se muestran los ficheros fuentes utilizados en el diseño y las dependencias entre ellos. También es aquí donde se elige el tipo de dispositivo donde se desea almacenar el diseño. Esta ventana posee diversas solapas para visualizar diferentes tipos de información relativa a las fuentes de diseño empleadas.
2. Ventana de Procesos. Esta ventana muestra todos los procesos necesarios para la ejecución de cada etapa de diseño. La lista de procesos se modifica dinámicamente dependiendo del tipo de fuente seleccionado en la ventana de ficheros fuente.
3. Ventanas de edición. Al hacer doble clic sobre un fichero fuente de la ventana de ficheros fuente se abre una ventana de edición para modificar el fichero (en caso de lenguaje VHDL), o bien se ejecuta el programa que permite editar el diseño (en caso de diseños esquemáticos o máquinas de estado).
4. Ventana de información, situada en la parte inferior. Muestra mensajes de error, aviso o información emitidos por la ejecución de los programas de compilación, implementación, etc.

Tanto en la ventana de procesos como en la de ficheros fuente es posible modificar las opciones de cada elemento a través del botón derecho del ratón, o bien a través de los menús del entorno de diseño; estos menús se modifican dependiendo del tipo de selección realizada en las ventanas de ficheros fuente y de procesos. Cada elemento mostrado en la ventana posee un icono diferente dependiendo del tipo de acción o fichero de que se trate, por ejemplo, nos indica si un elemento es un documento de texto, una acción para ejecutar en el entorno ISE o una acción para ejecutar por un programa adicional como puede ser ModelSim a la hora de simular. También muestra información sobre el estado que ha dado resultado tras la ejecución del proceso, es decir, si ha sido satisfactorio, si ha tenido errores, o ha tenido avisos. Las imágenes de la figura 3 muestran un ejemplo de los diferentes tipos de información mostrados en estas dos ventanas. Para la ventana de ficheros fuente, se indica si un fichero es de código, de vectores de test, si es un paquete, o una selección de dispositivo.

Figura 3. Detalle de las ventanas de ficheros fuente y de procesos

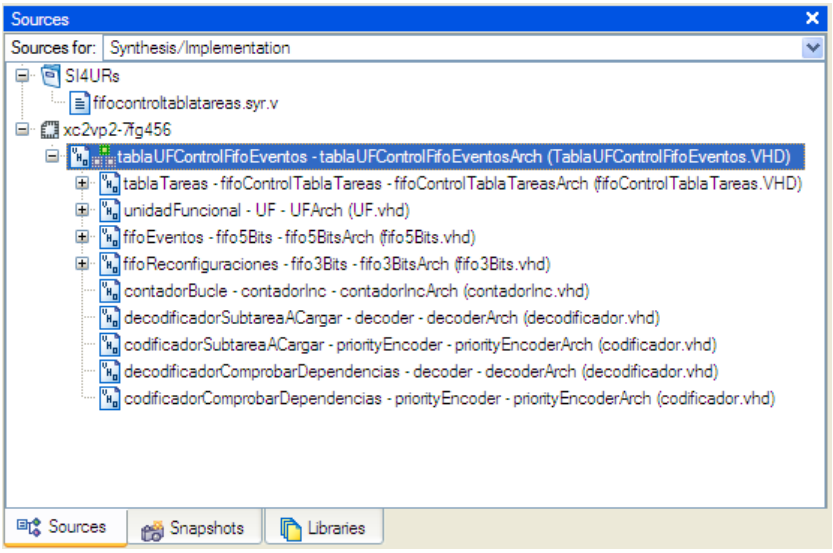
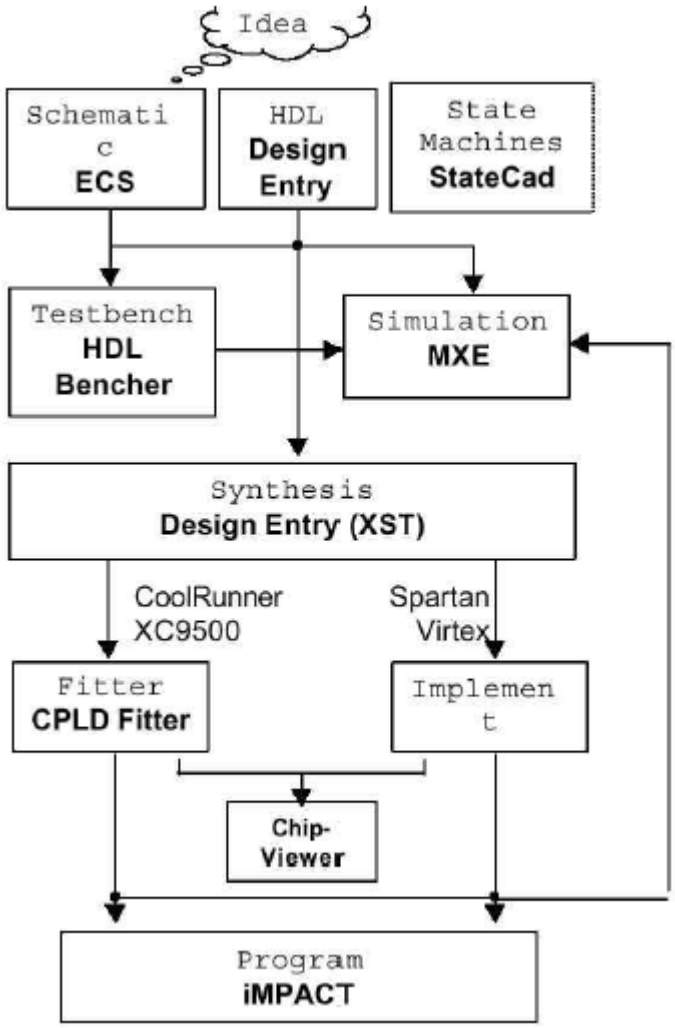


Figura 4. Proceso de diseño en Xilinx ISE



En concreto, la ventana de procesos incorpora todas las opciones necesarias para realizar todos los pasos de implementación de sistemas en lógica programable, incluyendo la edición y

verificación. La figura 4 muestra el diagrama de flujo de diseño en Xilinx ISE, y la figura 5 muestra las diversas partes en que se divide la ventana de procesos dependiendo de la tarea a realizar.

Figura 5. División de tareas dentro de la ventana de procesos

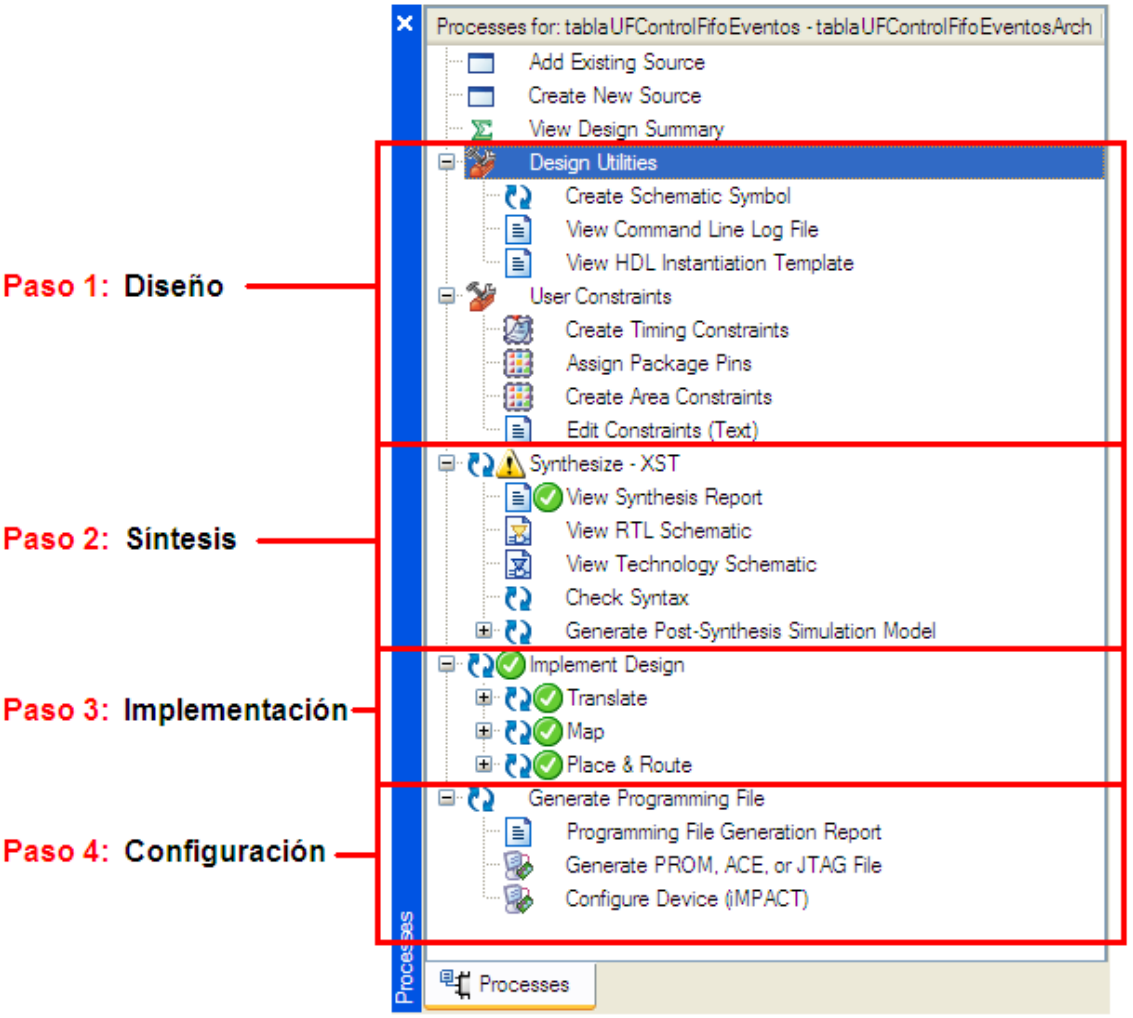
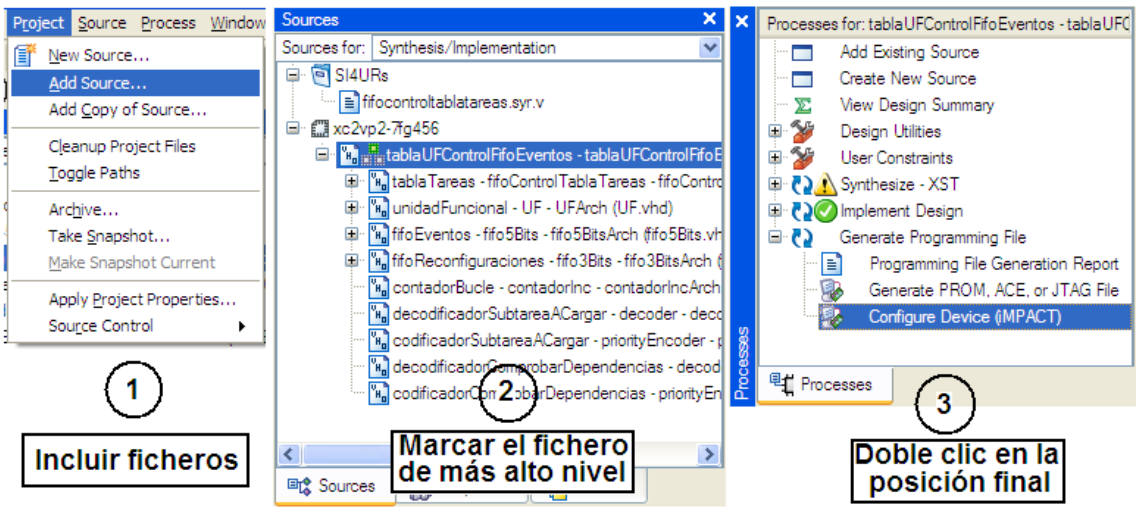


Figura 6. Proceso simplificado para el desarrollo de un diseño en Xilinx ISE



De manera resumida, el proceso de diseño resulta sencillo y se realiza en tres pasos, el primero consiste en añadir los ficheros fuente, en el segundo paso se selecciona el fichero de más alto nivel que se quiere implementar, y finalmente se hace doble clic sobre el último proceso al que se desea llegar, de este modo se ejecutarán todos los procesos intermedios necesarios para llegar al proceso seleccionado en último lugar (figura 6).

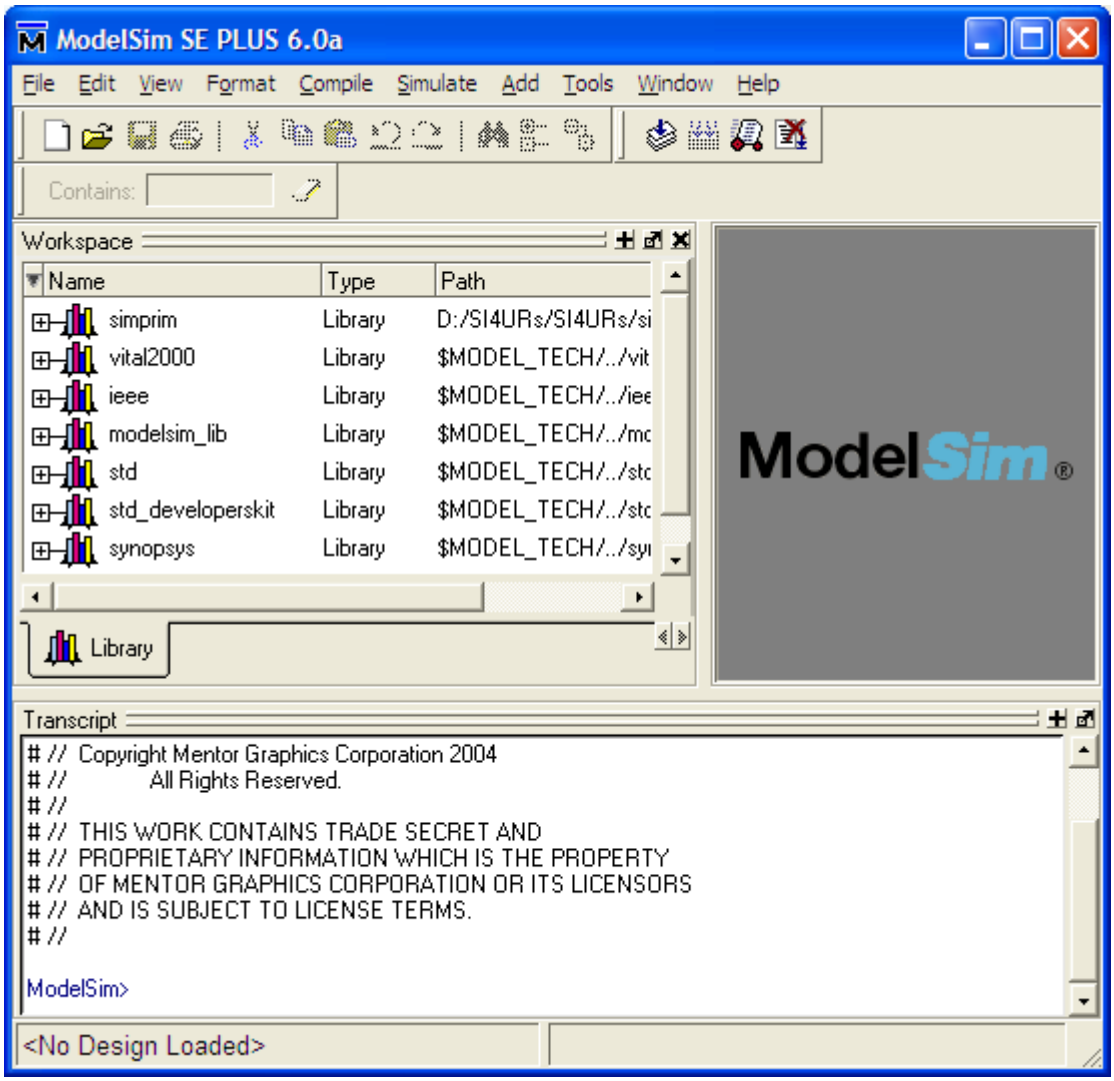
En el proyecto, este software se utilizó para la implementación en VHDL del gestor y para el testeo de los módulos básicos, con el propio simulador que incorpora ISE.

ModelSim 6.0a

Los bancos de prueba (o *testbenchs*) son una parte esencial del proceso de diseño, ya que permiten comprobar su correcto funcionamiento y ayudan en la automatización del proceso de verificación del diseño. Recoger estadísticas de la cobertura del código durante la simulación ayuda a asegurar la calidad y la minuciosidad de las pruebas.

El software ModelSim SE 6.0 utilizado en este proyecto, proporciona un entorno integrado de depuración (*Integrated Debug Environment*) que facilita el depurado eficiente de diseños basados en FPGAs, programados en cualquiera de los tres lenguajes siguientes: VHDL, Verilog y SystemC.

Figura 7. Ventana principal del simulador ModelSim SE 6.0



Además de simular el funcionamiento del sistema diseñado, el ModelSim SE 6.0 permite visualizar las señales de cada puerto del bloque simulado y realizar las modificaciones necesarias del código en función del resultado obtenido.

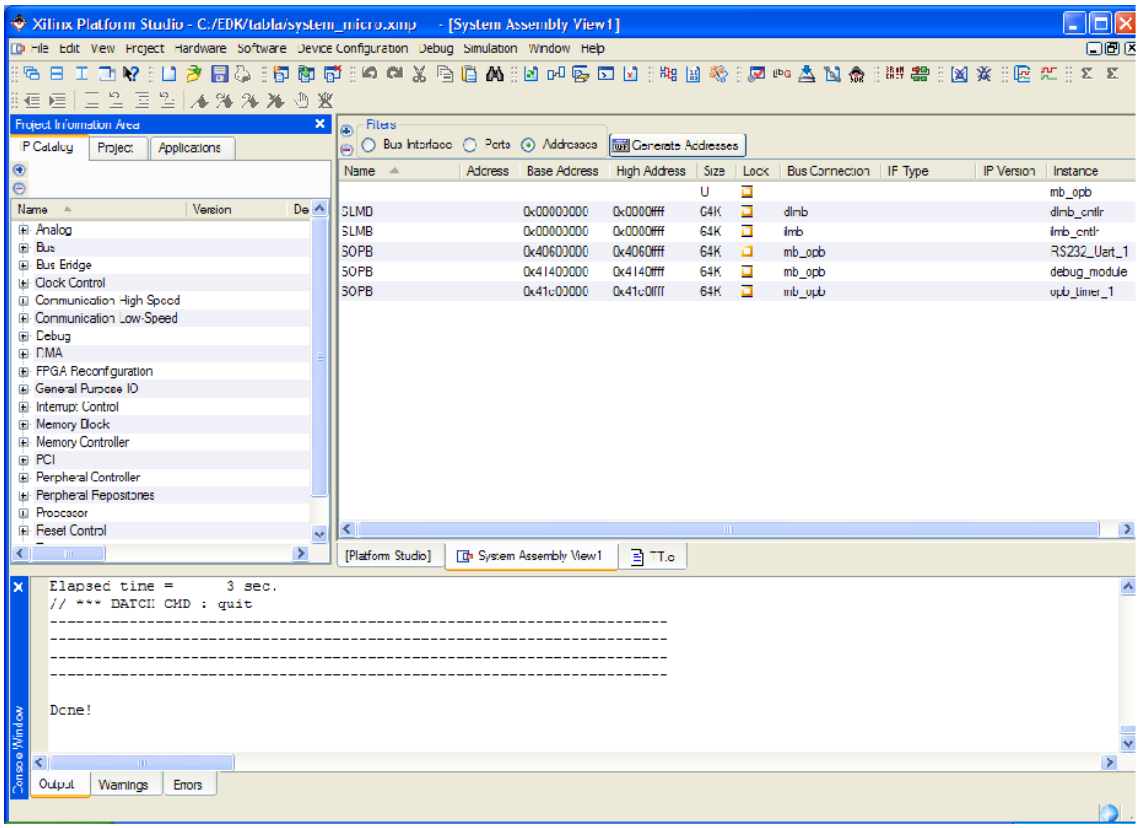
Partiendo del código proporcionado por el programa Xilinx ISE 9.1i, la simulación se puede realizar a distintos niveles: desde el nivel más alto en el que se utilizan modelos “ideales” de los componentes hasta el nivel Post Place & Route donde cada componente ha sido mapeado a un elemento concreto de la FPGA y posteiormente han sido conectadas entre sí, de forma que la simulación se basa en modelos muy precisos de los componentes que incluso incluyen los retardos de las conexiones.

Xilinx EDK 9.1i

Xilinx EDK es la abreviatura de Xilinx Embedded Development Kit y como indica la palabra es un conjunto de herramientas para el diseño y desarrollo de sistemas embebidos. Esta herramienta de diseño se encuentra dividida en tres subbloques o subpestañas principales. En el primer de ellos, encontramos un catálogo con todos los elementos de que consta nuestro diseño “IP Catalog”; el segundo subbloque consta de la información del projecto “Project” y el tercer y último subbloque “Applications”, muestra las aplicaciones (en código C o C++) asociadas a nuestro diseño.

Asimismo contamos con la barra de herramientas y de iconos para generar el fichero .bit del proyecto, guardar, cargar, descargar dicho fichero a la fpga y muchas más opciones.

Figura 8. Xilinx EDK



RECONFIGURACIÓN DINÁMICA

Opciones arquitectónicas

El HW reconfigurable ha captado la atención de un amplio número de grupos de investigación desde su aparición. Está ganando popularidad tanto a nivel académico, donde cada vez son más los grupos de investigación dedicados a su estudio, como a nivel comercial. Este considerable desarrollo ha propiciado que existan numerosas opciones arquitectónicas por las que hay que decantarse a la hora de realizar un diseño HW que convendrá comentar brevemente. En particular, podemos optar por los siguientes criterios:

1) ***Grano fino o grano grueso.*** Las arquitecturas de grano fino permiten cambiar la configuración para alterar el comportamiento de la placa a nivel de bit. Por otro lado, las arquitecturas de grano grueso trabajan con palabras de varios bits; partiendo del hecho de que las operaciones se realizan a nivel de palabra. El grano fino proporciona máxima flexibilidad para crear una configuración óptima de un algoritmo, a cambio de pagar un precio en área, longitud y cantidad de conexiones existentes, tiempo de reconfiguración, consumo de energía e incluso el rendimiento. El grano grueso representa el extremo opuesto.

2) ***Utilización de uno o múltiples contextos***. Una FPGA de un contexto es aquélla que sólo guarda la información de una reconfiguración en un instante de tiempo dado. Por otro lado, la utilización de múltiples contextos implica que la plataforma dispone de la memoria suficiente como para guardar múltiples configuraciones, así como la capacidad de cambiar de uno a otro en tiempo de ejecución cada vez que sea necesario. La inmensa mayoría de las FPGAs del mercado sólo soportan un único contexto, debido al sobrecoste de área de almacenamiento de los diferentes contextos, así como la lógica necesaria para la selección de los mismos.

3) ***Reconfiguración parcial o global.*** Hablamos de reconfiguración global cuando las reconfiguraciones de la FPGA se han de realizar simultáneamente en toda la placa. Por otro lado, cuando se permite que una parte de la plataforma pueda cambiar su funcionalidad mientras el resto permanece inalterado se habla de reconfiguración parcial. Esta última tiene dos ventajas importantes: el tiempo de reconfiguración es menor (al ser sólo parte del HW de la placa el involucrado) y permite gestionar de forma independiente la ejecución de múltiples tareas.

Para el desarrollo de este proyecto nos hemos decantado por un modelo de grano fino combinado con algunos componentes empotrados dentro de la FPGA, al representar un compromiso entre consumo de recursos y rendimiento. Utilizamos un único contexto, ya que las FPGAs de que disponíamos sólo operaban de ese modo. Finalmente, hacemos uso de reconfiguración parcial, elección clave para que se pueda desarrollar un sistema que gestione la reconfiguración y la ejecución de tareas en varias unidades reconfigurables y/o varios procesadores.

Hacia un sistema con varias unidades reconfigurables

En la planificación típica de un sistema operativo, la existencia de muchos procesos y/o *threads* puede llegar a degradar considerablemente el rendimiento del sistema. Como posibles soluciones a este problema, se pueden plantear, fundamentalmente, dos mejoras: aumentar el número de procesadores y migrar tareas SW a HW. Ambas propuestas buscan paralelizar a nivel de datos y a nivel de cómputo.

El HW dinámicamente reconfigurable permite llevar a cabo la primera propuesta (ejecución SW/HW), cuyas ventajas e inconvenientes son:

Ventajas: Se realiza una ejecución totalmente en paralelo y permite combinar de manera razonable una ejecución rápida con consumo de energía aceptable.

Inconvenientes: No es trivial pasar de SW a HW, así como tampoco lo es reconfigurar parcialmente y en tiempo de ejecución parte de la FPGA. Asimismo, las latencias de reconfiguración pueden degradar el rendimiento del sistema. Esto exite una buena planificación que las oculte; y estas planificaciones suelen ser costosas: A menudo hay que encontrar una planificación casi óptima para que merezca la pena llevarla a cabo y se manejan estructuras de datos complejas, como listas, grafos…

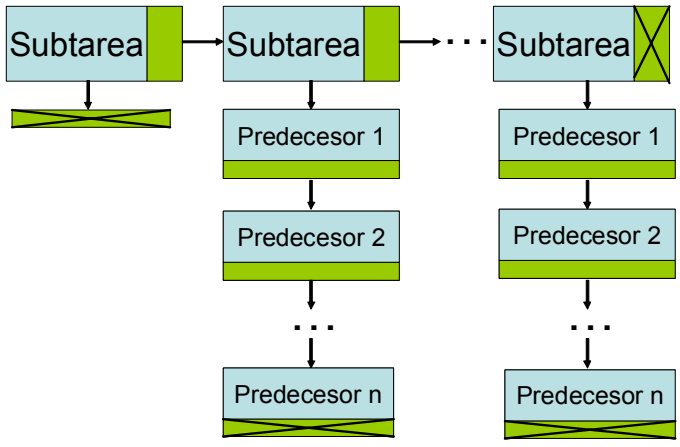
Quizá las características más interesantes del HW dinámicamente reconfigurable sean su versatilidad y el hecho de que se puedan ejecutar tareas totalmente en paralelo manteniendo un compromiso entre tiempo de reconfiguración y/o ejecución y consumo de energía. Este motivo, unido a que existe un amplio trabajo de investigación que queda aún por realizar en este campo (la gran parte de las posibilidades que ofrece el HW dinámicamente reconfigurable están todavía desaprovechadas debido a la falta de soporte que proporcionan los fabricantes) nos ha parecido suficiente como para habernos decantado por la realización de una labor de investigación propia en este sentido, así como en el desarrollo de un gestor que permita la ejecución de tareas en una plataforma en HW dinámicamente reconfigurable siguiendo una planificación dada.

EJEMPLO DE MOTIVACIÓN

En co-diseño HW/SW con frecuencia las tareas se representan como grafos de subtareas, donde una subtarea es la unidad básica de planificación, que representa un núcleo con carga computacional significativa (uno o varios bucles importantes) que pueden ser asignado a una unidad reconfigurable, y las aristas del grafo representan las dependencias entre subtareas. Para controlar la ejecución de estas tareas, el procesador debe manejar estructuras de datos complejas en tiempo de ejecución, con una gran carga computacional. Por ejemplo, la figura 9 muestra la representación de un grafo de subtareas usando una lista enlazada. En este caso la lista principal incluye todas las subtareas del grafo, y cada nodo contiene otra lista con todos sus predecesores. Si el planificador tiene que comprobar si una tarea dada está lista para su ejecución, debe buscar primero el nodo correspondiente en la lista, y luego leer el número de predecesores. Después de esto, debe recorrer la lista de predecesores entera, y mirar el estado de todos ellos para comprobar si ya terminaron su ejecución. La complejidad de esta operación es de $\Theta(N^2)$, donde N es el número de subtareas en el grafo.

Por supuesto, hay muchas maneras de mejorar la eficacia de esta operación, como la utilización de tablas indexadas para permitir un acceso directo a cada nodo del grafo en lugar de seguir los punteros, o actualizar la lista de predecesores cada vez que una subtarea termine su ejecución. Sin embargo, el manejo de datos complejos será necesario en cualquier caso.

Figura 9. Grafo de subtareas utilizando una lista enlazada



Además, dado que el SO y el planificador deben interactuar con las URs, frecuentemente

se realizarán comunicaciones HW/SW. Esta comunicación implicará probablemente manejo de interrupciones, y utilizar un bus compartido que con frecuencia es el cuello de botella del sistema.

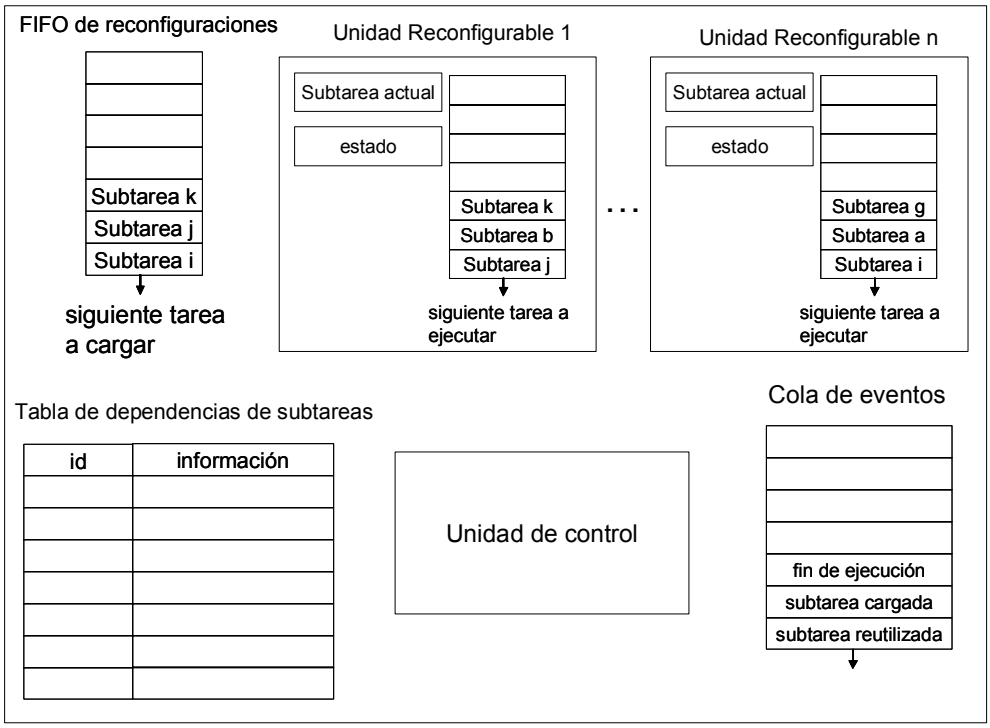
Nuestro gestor proporciona dos ventajas principales para reducir este problema. En primer lugar, el gestor controla directamente la ejecución de un grafo de subtareas en las URs, y gracias a su tabla asociativa operaciones como la actualización del grafo de subtareas, o la comprobación de si una subtarea está lista para ejecución se realizan rápidamente: un sólo ciclo de reloj para actualizar cada dependencia y un solo ciclo de reloj para comprobar si una subtarea está lista. Segundo, nuestro gestor sólo tiene que comunicarse con el procesador dos veces durante la ejecución de un grafo, una vez al principio de la ejecución, para recibir toda la información, y otra vez al final, para informar al procesador que la ejecución se realizó correctamente.

DESARROLLO DEL PROYECTO

VISIÓN GENERAL DEL SISTEMA

Nuestro sistema es un gestor de ejecución de tareas de un procesador basado en HW reconfigurable. En la figura 10 se puede observar la organización del gestor y seguidamente se procederá a la descripción de cada una de sus partes.

Figura 10. Esquema del gestor



Sus componentes son:

- **Tabla de dependencias entre subtareas:** Cuando se debe ejecutar una nueva tarea (representada mediante un grafo) las dependencias de cada subtarea se almacenan en una tabla asociativa. Cada vez que una subtarea finaliza su ejecución, todos sus sucesores se actualizan, y esa tarea se elimina de la tabla. Esta tabla se consulta antes de comenzar a ejecutar una subtarea para comprobar si todas las dependencias están resueltas, es decir, si las subtareas predecesoras se han ejecutado ya. Esta operación se puede realizar en un sólo ciclo de reloj.

- **Información de la unidad:** El gestor asigna una FIFO y dos registros a cada unidad reconfigurable. La FIFO almacena las subtareas asignadas a esa unidad siguiendo la planificación dada, mientras que los dos registros almacenan el estado de la unidad y la subtarea que en ese momento allí esté cargada, respectivamente.

- **FIFO de reconfiguraciones:** Esta FIFO almacena la secuencia de subtareas que deben ser reconfiguradas. Esta FIFO sólo se usa si estamos aplicando la *aproximación basada en una planificación.*

- **Cola de eventos:** Almacena los eventos que se generan en tiempo de ejecución, como el fin de la ejecución de una subtarea o el fin de una reconfiguración.

- **Unidad de control:** Esta unidad lee los eventos para identificar si una subtarea puede comenzar su ejecución o si es posible anticipar una configuración.

El sistema es escalable en cuanto a número de unidades reconfigurables, número máximo de predecesores y/o sucesores que puede tener una subtarea y longitud de los identificadores de las subtareas. Esto nos ha permitido hacer pruebas con diferentes valores para estos parámetros y así determinar cuáles de estos valores son los más adecuados para un funcionamiento óptimo.

DESCRIPCIÓN DEL SISTEMA

Nuestras FIFOs

1) Las BLOCK RAMs

En primer lugar, mencionar que, al ser las FIFOs dispositivos de almacenamiento masivo de datos, es muy probable que si no hacemos un uso eficiente de los recursos disponibles en la FPGA, nuestro diseño ocupará demasiada área de integración y será poco eficiente.

Para subsanar este problema utilizamos un componente de memoria de Xilinx llamado *RAMB16Sn*, en donde n puede ser: 1, 2, 4, 9, 18 y 36. No todas las FPGAs los tienen; sin embargo, sólo hay que consultar la tabla 1 para asegurarnos de que en una Virtex XC2VP30-5 FG676C, que es la placa que utilizaremos, este componente está disponible:

Tabla 1. Disponibilidad del módulo RAMB16Sn en distintos modelos de FPGA

Spartan-II, Spartan-IIE	Spartan-3	Virtex, Virtex-E	Virtex-II, Virtex-II Pro, Virtex-II Pro X	XC9500/XV/XL	CoolRunner XPLA3	CoolRunner-II
N/A	Primitive	N/A	Primitive	N/A	N/A	N/A

Los componentes *RAMB16_S1, RAMB16_S2, RAMB16_S4, RAMB16_S9, RAMB16_S18* y *RAMB16_S36* son bloques de memoria de acceso aleatorio con escritura asíncrona. Tienen 16384 bits de memoria para datos. Por su parte, el *RAMB16_S9, RAMB16_S18* y el *RAMB16_S36* tienen además 2048 bits adicionales de paridad que no utilizaremos. Las posibles configuraciones del *RAMB16_Sn* son las que se pueden observar en la tabla 2, mientras que su comportamiento es el que se puede observar en la tabla 3.

Tabla 2. Posibles configuraciones del módulo RAMB16Sn

Componente	Celdas de datos		Celdas de paridad		Bus de direcciones	Bus de datos	Bus de paridad
	Profundidad	Anchura	Profundidad	Anchura			
RAMB16_S1	16384	1	-	-	(13:0)	(0:0)	-
RAMB16_S2	8192	2	-	-	(12:0)	(1:0)	-
RAMB16_S4	4096	4	-	-	(11:0)	(3:0)	-
RAMB16_S9	2048	8	2048	1	(10:0)	(7:0)	(0:0)
RAMB16_S18	1024	16	1024	2	(9:0)	(15:0)	(1:0)
RAMB16_S36	512	32	512	4	(8:0)	(31:0)	(3:0)

Tabla 3. Comportamiento del módulo RAMB16Sn

Entradas								Salidas			
GSR	EN	SSR	WE	CLK	ADDR	DI	DIP	DO	DOP	Contenido de la RAM	
										Datos en la RAM	Paridad en la RAM
1	X	X	X	X	X	X	X	INIT	INIT	S/C	S/C
0	0	X	X	X	X	X	X	S/C	S/C	S/C	S/C
0	1	1	0	↑	X	X	X	SRVAL	SRVAL	S/C	S/C
0	1	1	1	↑	dir	datos	pdatos	SRVAL	SRVAL	RAM(dir) =>datos	RAM(dir) =>pdatos
0	1	0	0	↑	dir	X	X	RAM(dir)	RAM(dir)	S/C	S/C
0	1	0	1	↑	dir	datos	pdatos	S/C [a] RAM (dir)[b] datos[c]	S/C [a] RAM(dir)[b] pdatos[c]	RAM(addr) =>datos	RAM(dir) =>pdatos

GSR=Señal para Set y Reset globales

INIT=Valor especificado en el atributo INIT para los datos en memoria. Su valor por defecto es a ceros
SRVAL= Valor de salida tras activarse SSR, que coincide con el valor del atributo SRVAL, especificado por el usuario.

S/C= Sin cambios
dir=dirección de la RAM
RAM(dir)=Contenidos de la RAM en la dirección DIR
data=Entrada de datos en la RAM
pdata=Datos de paridad en la RAM

[a]WRITE_MODE=NO_CHANGE
[b]WRITE_MODE=READ_FIRST
[c]WRITE_MODE=WRITE_FIRST

Se pueden utilizar los atributos *INIT_XX* para especificar valores iniciales de los contenidos en memoria de una *RAMB16*. Esta inicialización se establece para cada *RAMB16_Sn* por medio de 64 valores hexadecimales (desde *INIT_00* hasta *INIT_3F*), cada uno de los cuales representa 16384 bits. Paralelamente, se pueden usar también los atributos *INITP_XX* para especificar valores iniciales en la memoria de paridad. Esta nueva inicialización se puede llevar a cabo mediante 8 atributos iniciales (desde *INITP_00* hasta *INITP_07*), compuestos por 64 valores hexadecimales, que conforman un total de 2048 bits; a través de los puertos configurados para 9, 18 o 36 bits. Si estos atributos (*INIT_XX* o *INITP_XX*) se quedan sin especificar, se les asigna como valor por defecto a ceros. A las cadenas de bits incompletas se les añaden ceros a la izquierda hasta completar el total del bus.

En las *placas Spartan-3, Virtex-II, Virtex-II Pro, y Virtex-II Pro X*, cada bit del registro de salida se puede inicializar a 0 o a 1. Esta inicialización se puede realizar por medio de dos atributos: *INIT* y *SRVAL*. El atributo *INIT* especifica la salida de este registro de salida cuando la FPGA se enciende (*power on*), mientras que el atributo *SRVAL* especifica la salida cuando se activa la señal de entrada SSR. Sus valores por defecto son a ceros.

En cuanto al modo de escritura, sólo comentar brevemente que el atributo *WRITE_MODE* controla los contenidos y la salida de la *RAMB16_SX*. Por defecto, su valor es *WRITE_FIRST*. Esto implica que si se realiza una escritura, en la salida se muestran los datos de la entrada.

También se puede configurar el *WRITE_MODE* a *READ_FIRST* para que, cuando se realice una escritura, mostrar en la salida el contenido en memoria que había antes de que se realizase dicha escritura. Asimismo, se puede configurar el *WRITE_MODE* a *NO_CHANGE* para que, cuando se realice una escritura, no se observen cambios en la salida de datos de la RAM.

Por último, comentar que utilizaremos un tipo de *RAMB16_SX* u otro en función de las necesidades de la FIFO en cuestión, en términos de longitudes de palabra y/o número de entradas. Hemos descubierto que cuanto mayor tamaño de palabra tenga la memoria que utilicemos, más recursos consume (en todos los casos se utiliza una BLOCK RAM, pero a mayor tamaño de palabra, se realiza un mayor número de interconexiones, con el consecuente consumo de recursos). Por ello, hemos implementado un total de cuatro tipos de FIFOs diferentes, que utilizaremos en los siguientes contextos:

- ***FIFO3BITS*:** Utilizamos una *RAMB16_S4*, para la FIFO de reconfiguraciones.
- ***FIFO5BITS*:** Utilizamos una *RAMB16_S9,* para la FIFO de eventos.
- ***FIFO13BITS*:** Utilizamos una *RAMB16_S18*, en los módulos para las URs.
- ***FIFO30BITS*:** Utilizamos *RAMB16_S36*, en la tabla de tareas.

Así conseguimos hacer un uso muy eficiente de los recursos disponibles. Por otro lado, la configuración de los parámetros de inicialización de las BLOCK RAMs es la siguiente:

- ***WRITE_MODE*** = Configurado a NO_CHANGE.
- ***Datos de paridad***: No utilizados.
- ***Contenido inicial de las FIFOs***: a ceros.
- ***GSR***: No utilizada
- ***SSR***: Reset activo a baja.

2) Implementación de las FIFOs

El manejo de una planificación de tareas dada exige por parte del sistema el uso de complejas estructuras de datos, como listas enlazadas o indexadas. En la figura 9 se puede observar un ejemplo de motivación.

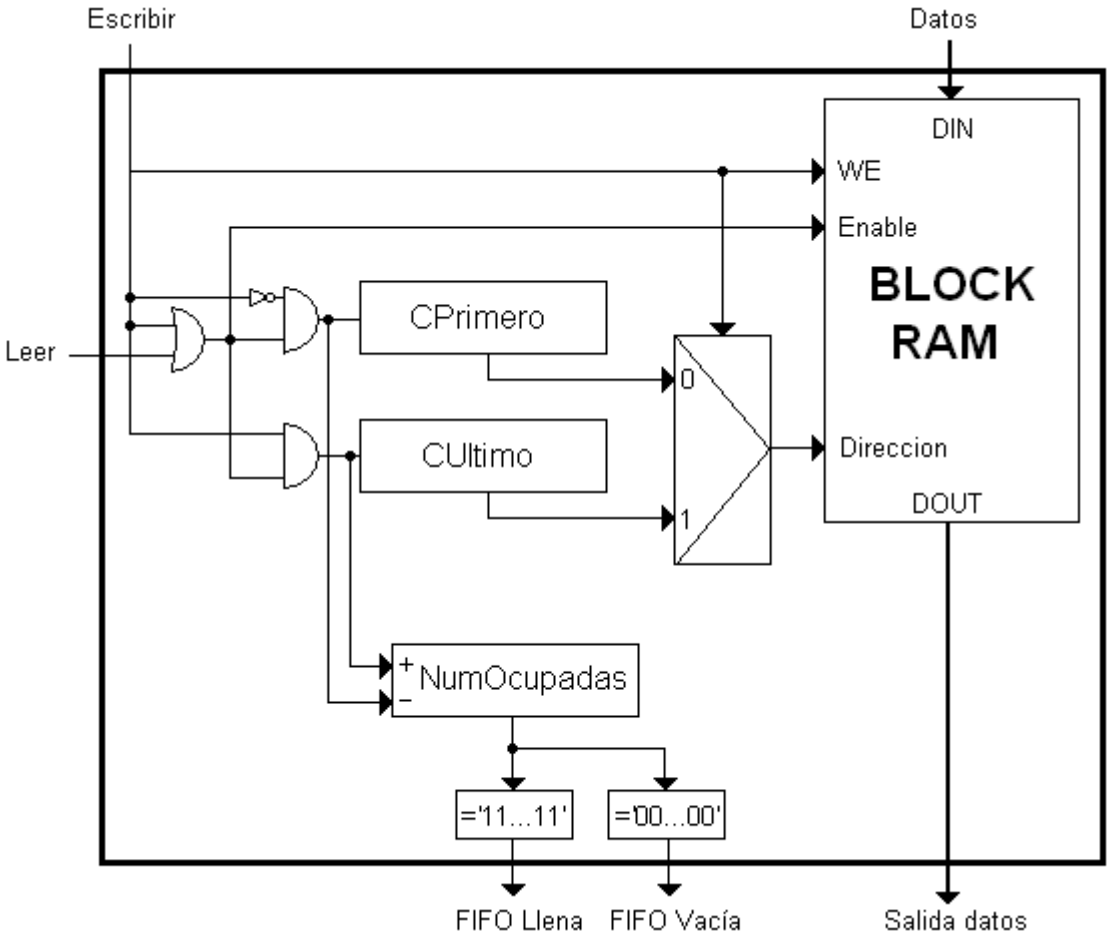
Esto nos ha obligado a implementar un HW que proporcione el soporte de almacenamiento necesario para llevar a cabo un manejo eficiente y transparente de operaciones de inserción, borrado y actualización de subtareas en listas enlazadas (figura 11). Este HW implementa la política de actualización FIFO (*First in, First out*), por lo que conceptualmente representa una cola cuyos elementos se insertan al final de la misma. Por otro lado, al realizar una operación de extracción de un elemento, se extrae el que esté en primer lugar en la cola. El número de entradas de las FIFOs es constante.

Las operaciones que soporta son las siguientes:

a) ***Inserción***: Se inserta un elemento al final de la cola. Si la FIFO estuviera llena, no se llevaría a cabo esta operación y se activaría la señal "*FIFO llena*".
b) ***Extracción***: Se extrae el primer elemento de la lista. Si la FIFO se vaciara como consecuencia de una extracción de su único elemento en la misma, se activaría la señal "*FIFO vacía*". Si la FIFO ya estuviera vacía y se intentase realizar una extracción, ésta no se lleva a cabo con éxito y la señal "*FIFO vacía*" permanece activa.

Por otro lado, nuestras FIFOs sólo tienen una entrada de datos, por lo que solamente se pueden realizar una lectura o escritura a la vez. No se permiten lecturas y escrituras simultáneas; y en el caso de que se intentase realizar dicha operación, se daría prioridad a las escrituras. (Hemos configurado el *WRITE_MODE* de la RAM a *NO_CHANGE*).

Figura 11. Esquema de las FIFOs desarrolladas



En cuanto a la implementación, su esquemático es el que aparece en la figura 11. Utilizamos dos contadores (*CPrimero* y *CUltimo*) para guardar la dirección de la primera y de la última posición de los datos en la tabla, respectivamente. Cuando se hace una operación de escritura, se escribe en la posición que indique el contador *CUltimo* (que apunta a la posición siguiente a la última ocupada) y éste se incrementa. Por otro lado, cuando se hace una operación de extracción, se realiza una lectura en la *BLOCK RAM* seleccionando como dirección la que indique *CPrimero* y se decrementa este contador. Así nos aseguramos que los datos están siempre en posiciones contiguas en la *BLOCK RAM*. También utilizamos un contador ascendente/descendente para saber cuántas posiciones hay ocupadas en la FIFO. (*NumOcupadas*). Si este contador está a cero, la FIFO estaría vacía, mientras que si este contador está a su máximo valor posible, la FIFO estaría llena.

La tabla asociativa

Es la tabla asociativa en donde se guarda la información de la(s) tarea(s) presentes en el sistema. Esta tabla se utiliza para supervisar las dependencias entre las subtareas de una tarea, conceptualmente representada mediante un grafo. Su tamaño es escalable en cuanto a número de entradas y número máximo de sucesores.

Las operaciones que soporta son: *inserción, borrado y actualización,* y *comprobación*, que pasamos a describir a continuación:

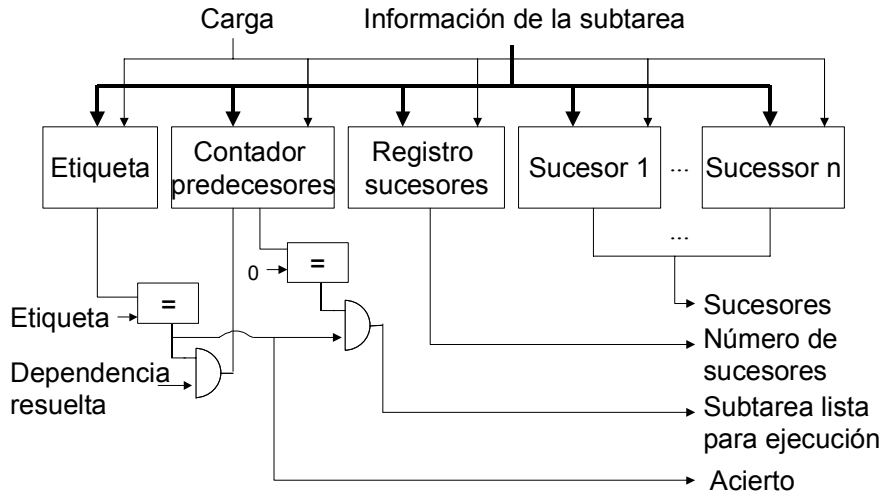
a) ***Inserción***: Esta operación escribe la información de la subtarea en la tabla. Como se trata de una tabla asociativa, la ubicación donde se escriben los datos no es relevante. Para seleccionar dónde se guarda una subtarea se utiliza la red iterativa, que siempre indica la primera entrada libre de la tabla; y se que genera un error si la tabla está llena. Con este enfoque la operación puede realizarse en un ciclo de reloj.
b) ***Borrado y actualización***: Esta operación se realiza cuando una subtarea termina su ejecución se elimina de la tabla, con lo que ésta puede ser usada en el futuro para ubicar otras subtareas. A continuación, se realiza una actualización de todas sus dependencias, consistente en disminuir en 1 el número de sucesores de cada uno de sus predecesores. Esto es posible gracias a que en la entrada de la tabla guardamos la información acerca de todos los identificadores de todos predecesores; con lo que la actualización consistirá en buscar cada uno de estos predecesores en la tabla y disminuir en 1 su número de sucesores. Queda claro, por tanto, que el objetivo de la actualización es garantizar en todo momento que la información que contiene la tabla sea correcta. Esta operación tiene una complejidad de Θ(N), donde N es el número de sucesores a actualizar.
c) ***Comprobación***: Esta operación consulta si una subtarea dada está lista para comenzar su ejecución, es decir, si están resueltas todas sus dependencias. Esto se realiza en un sólo ciclo de reloj, introduciendo la etiqueta de la subtarea como entrada de la tabla y leyendo la salida *subtarea preparada* en el siguiente ciclo. Una subtarea está preparada si su contador de predecesores es cero

Estructuralmente, la tabla está compuesta por una serie de entradas unidas entre sí mediante una red iterativa (que sirve para decidir en qué entrada se guardará una nueva subtarea que entre en la tabla). Asimismo, existe también un controlador que gestiona las distintas operaciones que la tabla permite realizar.

1) Entrada de la tabla

Consiste en un módulo básico en el que se guarda toda la información relacionada con una misma subtarea: su identificador, el número de predecesores, el número de sucesores, así como los identificadores de estos sucesores. En la figura 12 se puede observar su esquemático.

Figura 12. Entrada de la tabla de dependencias de subtareas



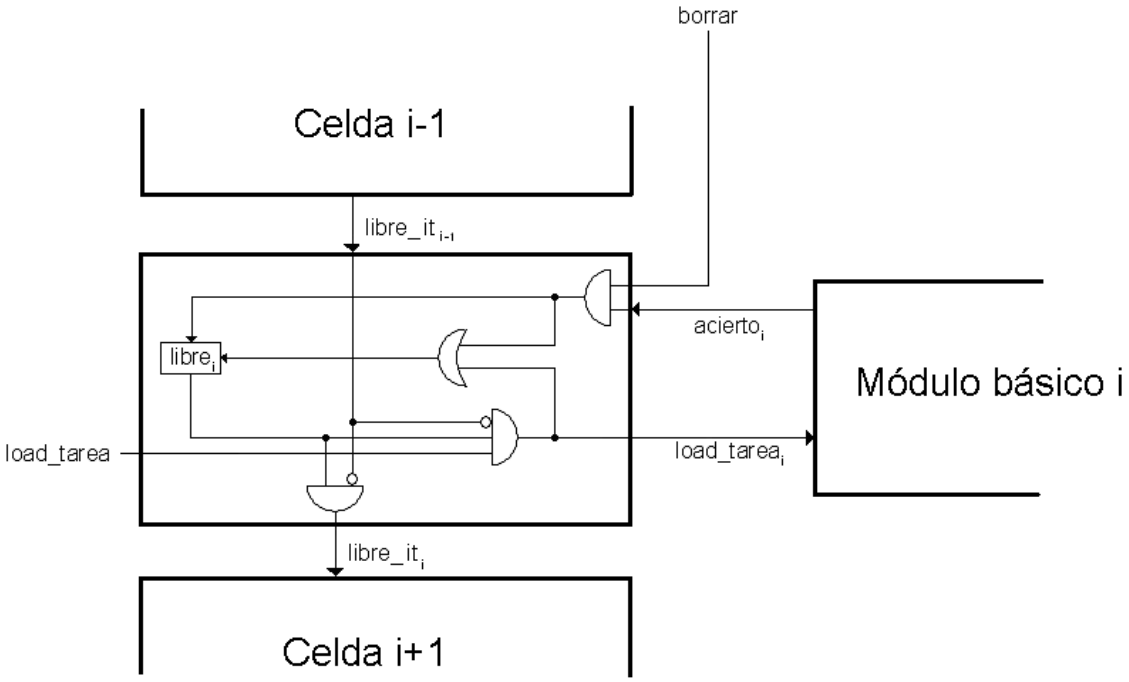
El registro "*etiqueta*" guarda el identificador de la subtarea que se encuentra en la entrada. El contador de predecesores y el registro de sucesores guardan la información acerca del número de predecesores y sucesores, respectivamente; y que será muy útil para gestionar las dependencias correctamente. Por otro lado, los registros "*Sucesor 1*", …, "*Sucesor n*" guardan los identificadores de los sucesores de la tarea que esté en esa entrada. El resto del HW sirve

para generar señales de control útiles para la correcta gestión de las dependencias: "*Subtarea lista para ejecución*" indica si la subtarea presente en la entrada está lista para su ejecución (equivalente a que su número de predecesores sea 0), y la señal "*acierto*" indica si la subtarea de la entrada es la que se pide desde la entrada "*Etiqueta*".

2) Red iterativa

Consiste en una red formada por una serie de celdas individuales conectadas entre sí y cada una de las cuales está a su vez conectada con una entrada de la tabla asociativa. En la figura 13 se puede observar su esquemático.

Figura 13. Red iterativa



Esta red se utiliza para saber cuál es la primera entrada de la tabla que se encuentra libre ("*load_tarea*$_i$"); y esta información es necesaria cuando se hace una inserción de una subtarea en la tabla.

Se trata de una red iterativa porque existe una señal (*libre_it*) que se van pasando unas celdas a otras. Su significado es el siguiente:

$$libre\_it_{i+1} = \begin{cases} 1 & si \;\; libre\_it_i = 0 \;\; AND \;\; \forall x \in [0..i-1] \, libre_x = 0 \\ 0 & en \;\; caso \;\; contrario \end{cases}$$

De esta forma, si hay varias posiciones libres en la tabla, sólo en la primera de estas celdas (empezando desde el subíndice 0) se activará *load_tarea*$_i$, por lo que nos aseguramos que una subtarea se cargará en la primera posición libre de la tabla.

Cada celda de esta red contiene un registro de un bit que indica si la entrada asociada a dicha celda está ocupada por una subtarea (1) o no (0). Por tanto, la actualización de su contenido sólo tendrá lugar en dos situaciones:

1) Cuando se cargue una nueva subtarea en el módulo básico asociado. Esto ocurre cuando la señal *load_tarea*$_i$ está activa, la celda está libre y la celda anterior no lo está. Si *load_tarea*$_i$ se activa, se desencadenaría la carga de la subtarea en el

módulo básico correspondiente y se activaría la señal de carga del registro. Lo que se carga en este caso en el registro es un 0, ya que la señal *borrar* no está activa.

2) Cuando se pretenda eliminar esa subtarea. Esto ocurre cuando la señal *borrar* está activa y cuando la subtarea que se quiere borrar está en el módulo básico asociado (*acierto*$_i$ es 1). Se pondrían a 1 simultáneamente la señal de carga del registro y su entrada de datos, y se cargaría un 1.

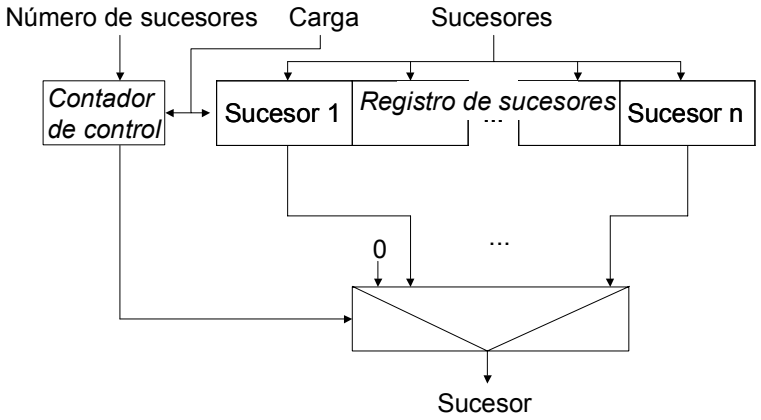
Las operaciones que implican un uso necesario del HW de la red iterativa tienen un tiempo de ejecución de un solo ciclo de reloj.

3) HW de control

Para que las operaciones que soporta la tabla se puedan llevar a cabo correctamente es necesario cierto HW de control. Este HW consta de:

a) Por un lado, una circuitería combinacional que garantiza que la operación de *borrado y actualización* se lleve a cabo correctamente. En la figura 14 se puede observar su estructura.

Figura 14. Soporte HW para la operación borrado y actualización



Disponemos de un contador, un registro de sucesores y un multiplexor. En el contador y en el registro de sucesores se guardan el número de sucesores y los identificadores de los sucesores que tiene la tarea cuyas dependencias queremos actualizar, respectivamente.

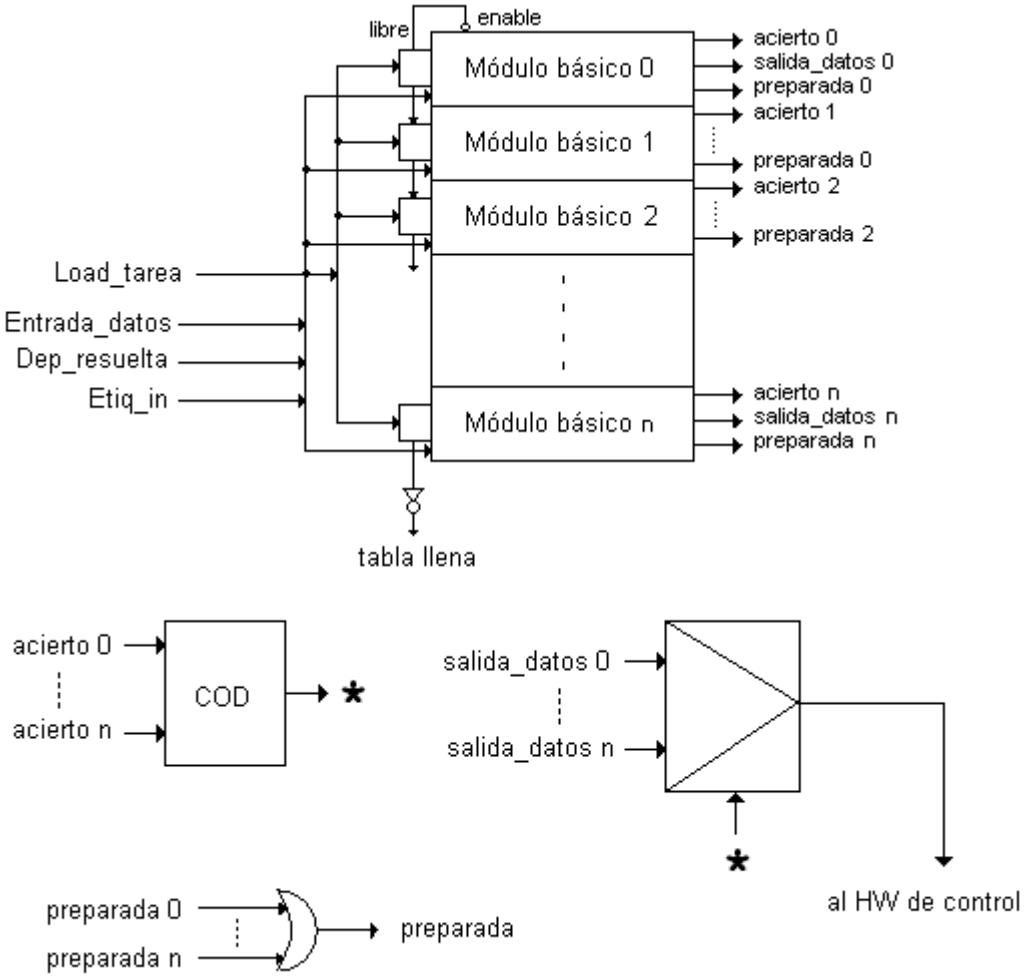
Su funcionamiento es el siguiente: se empieza cargando la información correspondiente en el contador y en el registro de sucesores. Después, se actualiza secuencialmente cada sucesor, operación que se realiza en un ciclo de reloj para cada uno. En cada ciclo se realizan tres operaciones: uno de los sucesores se selecciona en el registro de sucesores usando el *contador de control* y el multiplexor, se activa la entrada de la tabla asociativa *dependencia resuelta* y el *contador de control* se decrementa. Cuando el contador de control toma el valor cero la operación finaliza. Por su parte, cuando la tabla detecta que la señal *dependencia resuelta* está activa, decrementa el contador de predecesores de la subtarea correspondiente.

b) Por otro lado, tenemos un controlador que garantiza que las operaciones anteriormente mencionadas se lleven a cabo correctamente, gobernando las señales de control del HW combinacional de este módulo de manera apropiada.

4) Esquemático a alto nivel

Todos los módulos anteriormente explicados y debidamente interconectados componen la tabla de tareas, cuyo esquemático a alto nivel es el que indica la figura 15:

Figura 15. Tabla de tareas



Vemos que la red iterativa está estrechamente relacionada con las celdas básicas de la tabla asociativa, pues cada celda de la red está conectada con una entrada de la tabla. Las entradas "*Entrada_datos*", "*Dep_resuelta*" y "*Etiq_in*" entran directamente en las celdas básicas de la tabla asociativa, mientras que "*Load_tarea*" entra *también* en la red iterativa.

Por otro lado, el codificador sirve para determinar la posición en que se encuentra la celda cuyo contenido coincide con la entrada "*Etiq_in*". Posteriormente, mediante un multiplexor se selecciona la salida de datos correspondiente a la tarea seleccionada y estos datos se utilizan en el siguiente HW de control, cuya función es garantizar que se realice la actualización de las dependencias en cada uno de sus sucesores. Esta actualización se realizará en la operación "*borrado y actualización*", explicada con detalle en el apartado "*Operaciones*".

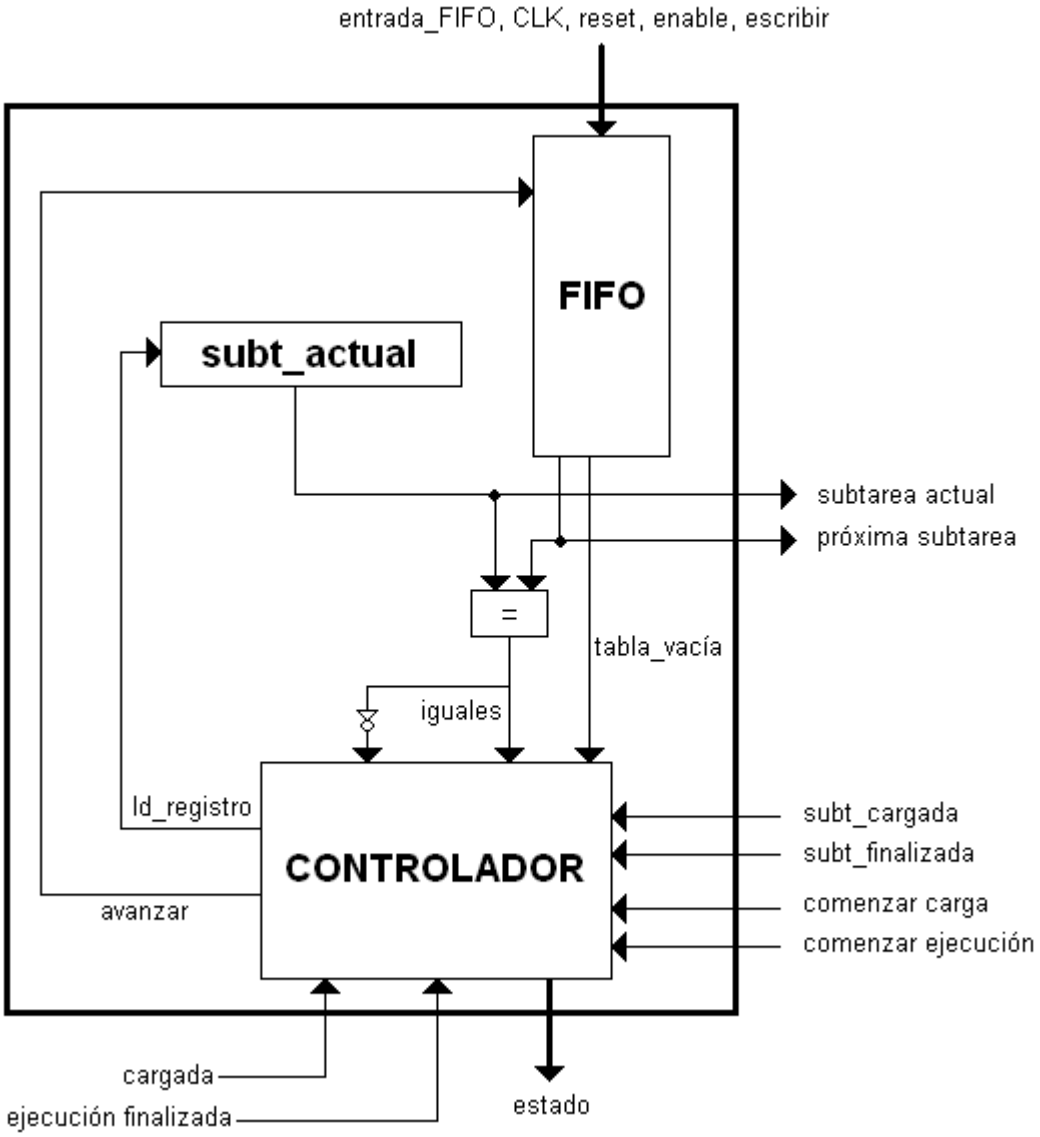
El módulo para las URs

Este módulo se encarga de caracterizar cada una de las unidades reconfigurables que del planificador. Su misión es proporcionar información a la unidad de control sobre el estado en el

que se encuentra cada una de las unidades reconfigurables. (Para más información acerca de este módulo consultar el apartado dedicado a la *unidad de control*.)

En el marco global del sistema, este módulo está continuamente funcionando y sincronizándose únicamente con el árbitro para gestionar las escrituras a la FIFO de eventos. A grandes rasgos, lo que hace es extraer subtareas de su FIFO local (no necesita ocuparse de cómo llegan ahí esas subtareas) y tratarlas secuencialmente. Para cada una de ellas, debe ocuparse de simular su reconfiguración y/o ejecución. Si no hay subtareas en su FIFO local, no hace nada.

Figura 16. Módulo para las unidades reconfigurables



Las tareas que realiza son:

1- Responder a las órdenes de comienzo de carga y comienzo de ejecución. En esos casos, este módulo simula la carga y/o la ejecución de la subtarea que se encuentre actualmente en la unidad.
2- Generar los eventos apropiados:
   a. Cuando se termina de ejecutar una subtarea se genera el evento “subtarea ejecutada”. El código de evento es: “10”

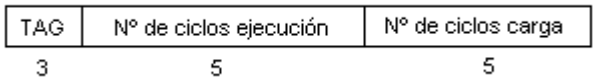
b. Cuando se termina de reconfigurar una subtarea o cuando una subtarea es reutilizada, el evento que se genera es “subtarea reconfigurada”. El código de evento es: “01”.

Estos eventos se escriben en la FIFO de eventos. Una situación problemática que podría suceder es que varias unidades reconfigurables intentasen escribir simultáneamente en dicha FIFO. Para que esa situación no represente problema alguno, existe un árbitro externo que garantiza el acceso exclusivo a las operaciones de escritura en la FIFO, a la vez que organiza las peticiones para que ninguna de ellas se quede sin atender. Para más información acerca del árbitro consultar el apartado dedicado al *árbitro*. Su esquemático es el que se puede ver en la figura 16.

Vemos que este módulo consta de los siguientes componentes HW:

1- Un registro en el que se guarda la subtarea actual de la unidad.
2- Una FIFO en la que se guardan las subtareas que están planificadas para ser ejecutadas en esa unidad. En concreto, usamos una *FIFO13BITS*, ya que las palabras que guarde tendrán el formato que se indica en la figura 17. Necesitamos saber cuántos ciclos tarda la ejecución y la reconfiguración de la tarea para poder así generar correctamente los eventos correspondientes.

Figura 17. Formato de las palabras en la FIFO de las URs



3- Un HW de control que gestiona los cambios de estado activando las señales correspondientes, así como las señales de salida del módulo. Este HW de control está compuesto por el comparador y el controlador, cuyo diagrama de estados se puede observar en la figura 18.

Este diagrama de estados es una máquina de Moore. Sus estados son los siguientes:

“***lee_fifo***”: La unidad reconfigurable está en este estado cuando está intentando leer de su FIFO de reconfiguraciones. Permanece en este estado mientras la tabla esté vacía. En el momento en que se detecte que hay al menos una subtarea en dicha FIFO, pasa al estado “*finalizada*”.

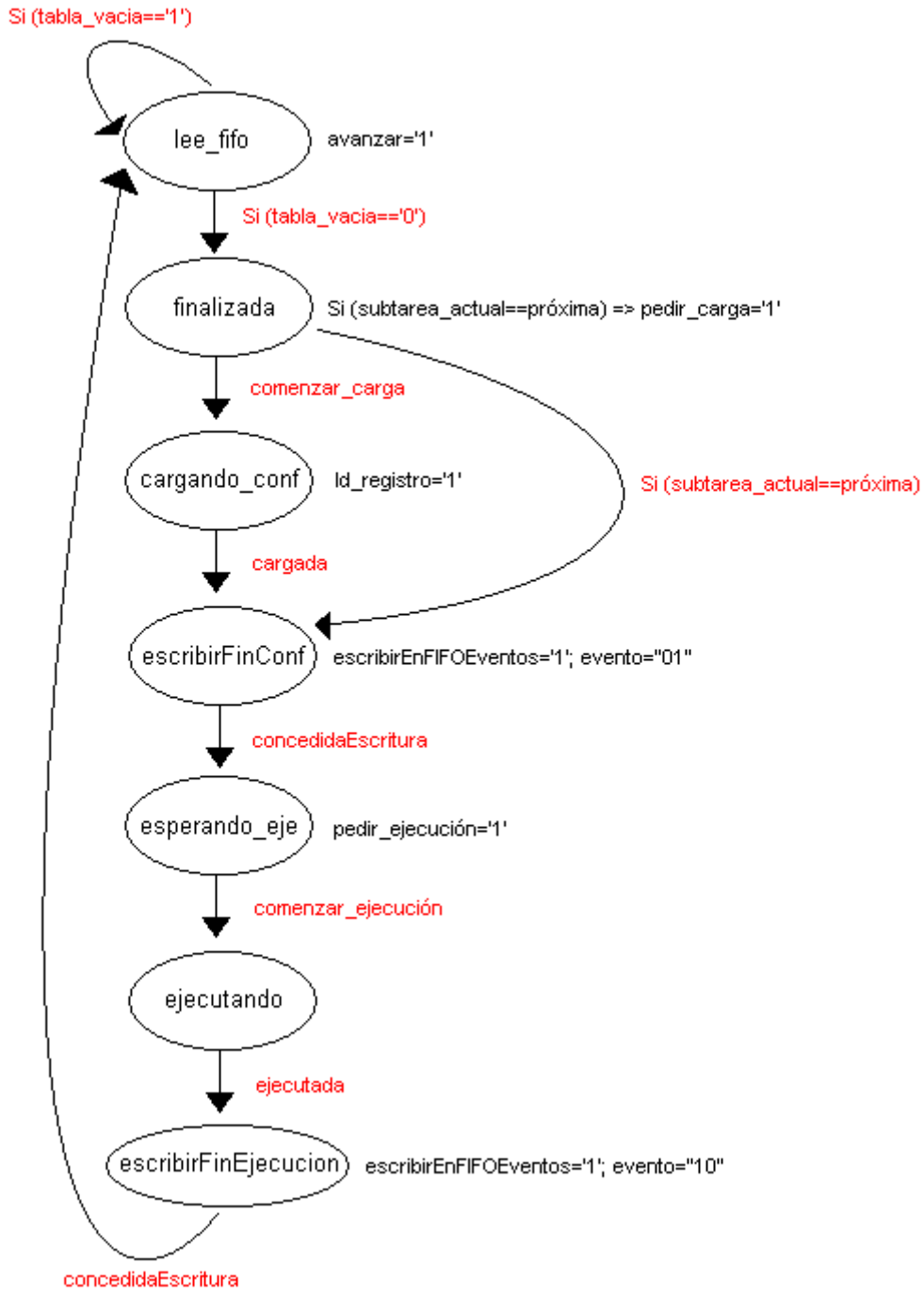
“***finalizada***”: La unidad reconfigurable está en este estado cuando se ha terminado de extraer una subtarea de la FIFO de reconfiguraciones. En este estado se decide si se debe o no pedir la carga de esta subtarea recién extraída (se debe reutilizar cuando es igual a la que había inmediatamente antes en la unidad). En caso de que haga falta reconfigurar (porque no coincida con la inmediatamente anterior) se pide la carga (*pedir_carga*==’1’) y, cuando se le ordene la carga (*comenzar_carga*), comience la reconfiguración. En caso contrario, pasa directamente al estado “*escribirFinConf*”.

“***cargando_conf***”: La unidad reconfigurable está en este estado mientras se esté llevando a cabo la reconfiguración de la subtarea. Es en este estado cuando se carga la subtarea en el registro de la unidad reconfigurable. El número de ciclos durante los que la unidad se encuentre en este estado dependerá del tiempo de reconfiguración, tiempo que viene incluido entre la información que se proporciona de la subtarea. Tan pronto como se recibe la confirmación de que la subtarea ha sido cargada (*cargada*==’1’) pasa al estado “*escribirFinConf*”.

“***escribirFinConf***”: La unidad reconfigurable está en este estado cuando se ha finalizado la reconfiguración de una subtarea o cuando se acaba de decidir que una subtarea debe ser reutilizada. En este estado se genera el evento “*subtarea reconfigurada*”. Este evento se genera tanto si una subtarea es reconfigurada como si es simplemente reutilizada. El motivo por el que se genere el mismo evento en dos situaciones diferentes es que ambas situaciones

son equivalentes desde el punto de vista del manejador de los eventos. La unidad permanece en este estado el tiempo que haga falta hasta que se le concede la escritura en la FIFO de eventos; porque, como ya se dijo anteriormente, puede ocurrir que varias unidades estén funcionando en paralelo y quieran escribir simultáneamente en dicha FIFO, con el consecuente problema de una escritura concurrente que la FIFO no soporta. Para garantizar el acceso exclusivo a estas operaciones de escritura debe existir un árbitro y es precisamente ese árbitro el que informa a los módulos UR acerca de las concesiones.

Figura 18. Diagrama de estados para el control de los módulos de las URs



"***esperando_eje***": La unidad reconfigurable está en este estado mientras se encuentre a la espera de comenzar la ejecución de su subtarea. Cuando se reciba desde el exterior la señal de que dicha ejecución debe comenzar, pasa al estado "*ejecutando*".

"***ejecutando***": La unidad reconfigurable está en este estado mientras la subtarea que contiene se esté ejecutando. Permanece en este estado tantos ciclos como tarde en llevarse a cabo dicha ejecución; y, cuando finaliza, activa la señal "*ejecutada*" y pasa al estado "*escribirFinEjecucion*".

"escribirFinEjecucion": La unidad reconfigurable está en este estado cuando se ha finalizado la ejecución de una subtarea. En este estado se genera el evento "*subtarea ejecutada*". Al igual que ocurría con el estado "*EscribirFinConf*", la unidad permanece en este estado el tiempo que haga falta hasta que se le concede la escritura del evento ahora generado en la FIFO de eventos. Cuando se recibe una confirmación de que dicho evento ha sido escrito con éxito (*concedidaEscritura*= ='1'), pasa al estado "*lee_fifo*".

La FIFO de reconfiguraciones

Disponemos de una FIFO en la que se guardan qué subtareas deben ser reconfiguradas en el sistema y en qué orden. Es, por tanto, en esta FIFO donde que se guarda la información acerca de la planificación que debe seguir el sistema. En concreto, utilizamos una *FIFO3bits*, ya que guardamos una lista de identificadores de subtareas; y dichos identificadores tienen 3 bits.

Esta FIFO se actualiza únicamente cuando entra una nueva tarea, momento en el cual se escriben en ella (en un determinado orden) todas las reconfiguraciones que han de producirse. Posteriormente, se irán extrayendo subtareas de esta FIFO y se les dará un tratamiento adecuado: buscar en qué unidad está; y cuando esta unidad esté en estado "finalizada", ordenar una ejecución (si tiene lugar una reutilización de esta subtarea en la UR) o una reconfiguración (en caso contrario). Este tratamiento tiene lugar en la unidad de control.

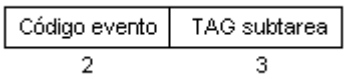
Sus características tecnológicas son las de una FIFO ya explicada anteriormente.

La FIFO de eventos

Disponemos de una FIFO de eventos para guardar y así poder gestionar los eventos que se producen en el sistema en tiempo de ejecución. En concreto, utilizamos una *FIFO5bits*, ya que el formato de palabras que se usarán en este caso es el que se indica en la figura 19.

Los eventos que se pueden producir en el sistema y, que, por tanto, son susceptibles de ser guardados aquí, se pueden generar desde los módulos que caracterizan a las URs o desde el cargador de tareas; este último caso sólo ocurre cuando entre una nueva tarea en el sistema.

Figura 19. Formato de las palabras en la FIFO de eventos



"nueva tarea": Este evento se genera cuando entra una nueva tarea en el sistema. Es el evento que menos veces se genera en el sistema; sólo una vez por tarea recibida. Su código binario es "11".

"fin de ejecución de la subtarea i": Este evento se genera cuando se termina de ejecutar una subtarea. Como una subtarea se ejecuta dentro de una UR, será el módulo que caracteriza a su UR el que genere este evento. Su código binario es "10".

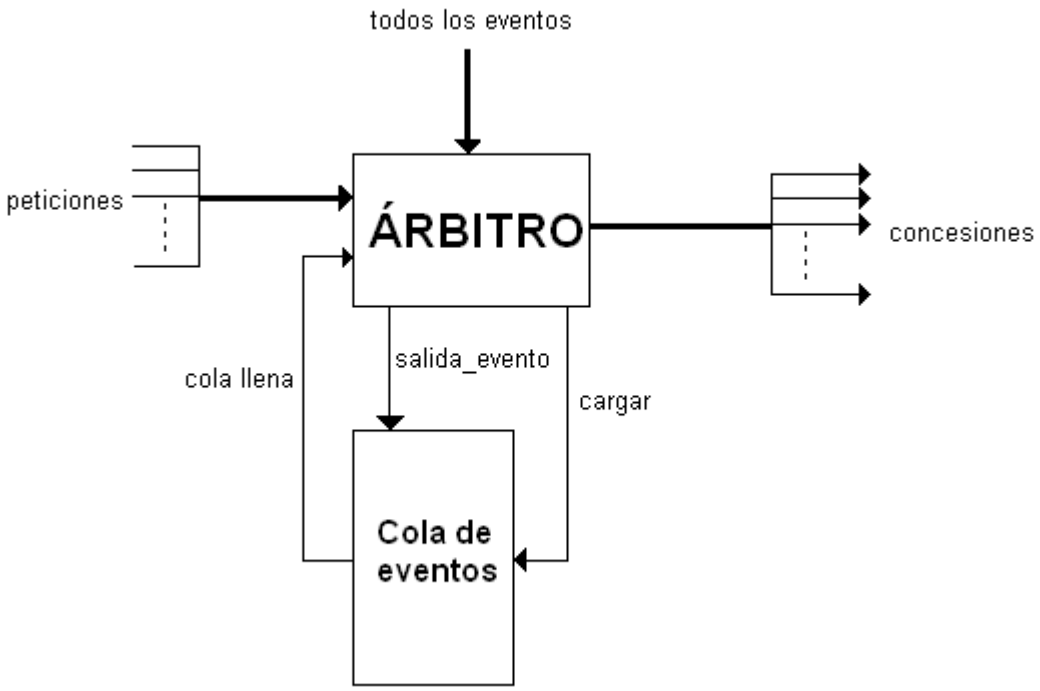
"fin de reconfiguración de la subtarea i": Este evento se genera cuando se termina de reconfigurar una subtarea. Se genera desde los módulos que caracterizan a las URs. Su código binario es "01".

"subtarea i reutilizada": Este evento se genera cuando se termina de ejecutar una subtarea. Se genera desde los módulos que caracterizan a las URs. A efectos prácticos, este evento y "*fin de ejecución de la subtarea i*" son equivalentes debido a que a ambos se les da el mismo tratamiento en la unidad de control. Por tanto, en las situaciones en las que se deba generar este evento, se generaría "*fin de reconfiguración de la subtarea i*", con la consecuente simplificación de la gestión de los eventos.

El árbitro

Acabamos de mencionar que en nuestro sistema se lleva a cabo una gestión de eventos, consistente en su generación, almacenamiento en una FIFO y posterior tratamiento. Debido a la naturaleza imprevisible (al menos desde el punto de vista del sistema) de estos eventos, es posible que se generen simultáneamente dos o más eventos y que, por tanto, se intenten escribir simultáneamente en la FIFO de eventos (entendemos "simultáneamente" por "en el mismo ciclo de reloj"). Esta es una situación problemática porque nuestras FIFOs no soportan dos o más escrituras simultáneas, al tener un solo puerto de escritura. Por tanto, es necesario garantizar la exclusión mutua en los accesos este puerto de escritura para que no quede ningún evento sin capturar. Para ello, hemos diseñado un árbitro (figura 20) que recibe todos los eventos que se generan en el sistema y establece una política de prioridades fijas de forma que se garantiza que sólo se intenta escribir en esta FIFO en un instante de tiempo dado.

Figura 20. Árbitro de interconexión con la FIFO de eventos



Este árbitro recibe todos los eventos en *eventos[1..n]*, (en el que *eventos[i]* es un código de dos bits) que se pueden producir en el sistema y *n* señales de peticiones (*peticiones[1..n]*) procedentes de cada uno de los módulos que los generan. Si *peticiones[i]* está activa, se hace una petición para que *eventos[i]* se escriba en la FIFO. Por otro lado, la señal de salida *concesiones[1..n]* indica las concesiones que se realizan: Si *concesiones[i]* está activa, se

permite la escritura de *eventos[i]* en la FIFO; en caso contrario, se prohíbe dicha escritura. Obviamente, sólo una de las *concesiones[i]* posibles estará activa en el mismo instante de tiempo.

El árbitro da la mayor prioridad a la escritura del evento "*nueva_tarea*"; seguido de los generados por las URs, de más significativa a menos significativa. (Se supone que las URs están identificadas por el árbitro de la forma *UR[1..n]*). Por tanto, su comportamiento es similar a un codificador de prioridad.

Además, el árbitro también controla las lecturas de la FIFO de eventos, dándoles mayor prioridad que a las escrituras.

La unidad de control

La unidad de control extrae los eventos de la cola y lleva a cabo las acciones apropiadas. Su pseudocódigo se describe en la figura 21.

Cuando un evento "*fin de ejecución*" se procesa, este módulo actualiza las dependencias guardadas en la tabla asociativa. Entonces, si el circuito de reconfiguración está libre, trata de empezar una reconfiguración. Si el sistema usa la aproximación voraz comenzará a leer los estados de las URs y si encuentra uno que esté ocioso y su FIFO de subtareas no esté vacía, comenzará el proceso de reconfiguración correspondiente. Si el sistema usa la opción basada en una planificación, leerá la FIFO de reconfiguración, y comprobará si es posible comenzar la reconfiguración correspondiente. Finalmente, la unidad de control comprobará si cualquiera de las subtareas que actualmente están cargadas puede comenzar su ejecución.

Para los eventos "*fin de reconfiguración*" y "*subtarea reutilizada*" la unidad de control comprobará si la subtarea que ha sido cargada puede comenzar su ejecución.

Finalmente, para el evento "*nuevo grafo*", la unidad de control leerá los datos desde el buffer de entrada y los almacenará en la tabla asociativa y en las respectivas FIFOs. Tras esto, si el circuito de reconfiguración está libre, comprobará si es posible comenzar a cargar una de las nuevas subtareas.

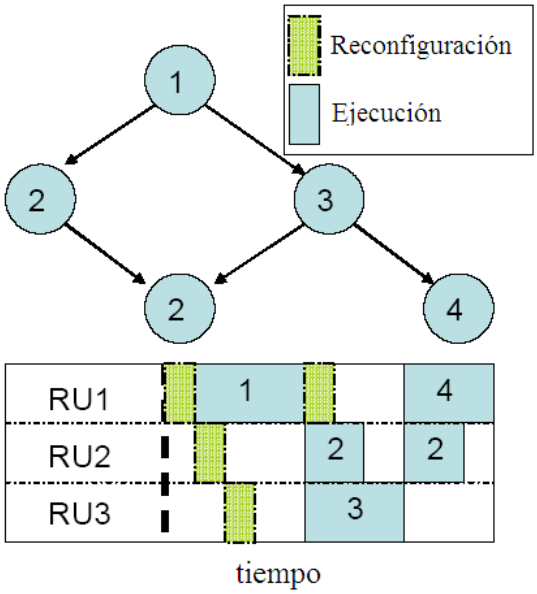
Figura 21. Pseudo-código de la unidad de control

```
Manejo de eventos:
 CASE evento IS:
  fin_de_ejecucion:
    actualizar_tabla
    buscar_una_reconfiguración()
    FOR i=0 hasta número de URs
       IF subtarea_preparada
            comenzar_ejecución()
  fin_de_reconfiguración;
    IF subtarea_preparada
        comenzar_ejecución()
        buscar_una_reconfiguración()
  nuevo_grafo:
    actualizar_tabla
    actualizar_FIFOs_UR
    IF politica_precarga=basado_planifica
           actualizar_FIFO_reconfig
           buscar_una_reconfiguración()
```

EJEMPLO DE EJECUCIÓN DE UNA TAREA

En esta sección describiremos paso a paso la ejecución del grafo de subtareas mostrado en la figura 22. Esta figura representa a su vez la planificación seleccionada.

Figura 22. Ejemplo de ejecución de un grafo de subtareas



Inicialmente, el procesador enviará la información referente al grafo y el método de planificación elegido a nuestro gestor. Cuando la información esté correctamente almacenada en un buffer de entrada, el procesador generará el evento *nuevo grafo*. Tan pronto como la unidad de control procese este evento, toda la información será almacenada en la tabla asociativa y en las FIFOs. Además, la unidad de control intentará empezar una reconfiguración. Si el sistema está usando una planificación *basada en prebúsqueda* y la secuencia de reconfiguración elegida por el planificador es: 1-2-3-4-2, la unidad de control comenzará la reconfiguración de la subtarea 1.

Cuando la subtarea 1 termine su reconfiguración, la UR1 generará un evento fin de reconfiguración.

Ahora, la unidad de control comprobará si la subtarea 1 puede empezar su ejecución. En este caso, no hay dependencias no resueltas, por lo tanto comenzará justo después de que la reconfiguración termine.

Además, la unidad de control comenzará la siguiente reconfiguración ya que el circuito de reconfiguración está libre y en la UR2 no hay ninguna tarea en ejecución.

Cuando esa reconfiguración termine, generará un nuevo evento fin de reconfiguración, y la unidad de control comenzará la reconfiguración de la subtarea 3. Además, se comprobará a su vez si la subtarea 2 puede comenzar su ejecución, pero en este caso la tabla asociativa indicará que hay una dependencia aún no resuelta.

Cuando esta reconfiguración finalice, un nuevo evento será procesado, pero esta vez, no dará comienzo una nueva reconfiguración, ya que las subtareas nunca serán reemplazadas hasta que hayan sido ejecutadas. Tampoco se comenzará la ejecución de ninguna subtarea, ya que las subtareas 2 y 3 tienen dependencias no resueltas.

Cuando la subtarea 1 finalice su ejecución se generará un nuevo evento, que actualizará las dependencias e intentará comenzar una reconfiguración, en este caso de la subtarea 4. Además se comenzará la ejecución de la subtarea 2.

Cuando la reconfiguración de la subtarea 4 finalice, se comprobará si puede comenzar su ejecución, pero en este caso no es posible ya que existen dos dependencias no resueltas. Además, se intentará planificar la reconfiguración de la subtarea 2, pero de nuevo no es posible porque la UR2 está ocupada.

El siguiente evento es el final de la ejecución de la subtarea 2. De nuevo, la tabla asociativa será actualizada y el sistema intentará comenzar una nueva reconfiguración, en este caso es posible comenzar la reconfiguración de la subtarea 2. Pero como esta tarea ya está cargada, el sistema la reutilizará, generando solamente un evento tarea reutilizada. Este evento se procesará a continuación, tratándose de comenzar la ejecución de la subtarea 2, pero dado que todavía existe una dependencia por resolver, la ejecución no podrá comenzar.

Cuando la subtarea 3 finalice, el sistema actualizará las dependencias e identificará que la subtarea 2 y 4 pueden empezar sus ejecuciones. Finalmente, cuando estas subtareas terminen, el gestor generará una interrupción para informar al procesador que la ejecución del grafo ha sido completada.

RESULTADOS EXPERMIENTALES

En esta sección, se evaluará el rendimiento de nuestro gestor y el área necesaria para implementarlo. Para este fin hemos implementado el circuito en una FPGA *VIRTEX-II PRO xc2rp30* utilizando *ISE 9.1i* como plataforma de desarrollo.

En nuestros diseños hemos empleado en todos ellos genéricos para que sean ajustables, como por ejemplo, el tamaño de la tabla asociativa, el máximo número de sucesores por tarea y el número de URs en el sistema. Puesto que estos parámetros influirán en el rendimiento y el coste del sistema, evaluaremos un pequeño gestor con una tabla asociativa de tan sólo ocho entradas y tres URs, y más adelante estudiaremos cómo evolucionan el rendimiento y el coste cuando el sistema aumenta de tamaño.

Como primer experimento hemos ejecutado nuestro gestor para planificar el grafo de subtareas de la figura 22, y algunos otros grafos de tareas correspondientes a tres aplicaciones multimedia: un decodificador JPEG, una aplicación de reconocimiento de patrones y un codificador MPEG. La tabla 4 muestra los resultados de esas tres ejecuciones. La columna de retardo añadido incluye todos los retrasos generados en la ejecución del grafo de subtareas debido a todos los cálculos realizados por nuestro gestor de ejecución.

Tabla 4. Evaluación del rendimiento

Tarea	Número de subtareas	Retardo añadido	Penalización
Figura 7	6	400 ns	-66%
JPEG	4	450 ns	-75%
Patrón rec.	7	420 ns	-80%
MPEG	5	580 ns	-70%

La columna penalización contiene el porcentaje de disminución de las penalizaciones generadas por las reconfiguraciones gracias a la técnica de prebúsqueda. Explicaremos los resultados tomando el grafo de la figura 22 como ejemplo. En este caso, sólo cuatro de los eventos serán manejados por el gestor han introducido retardos. Estos eventos y los retardos pueden ser visualizados en la tabla 5. Estos eventos han generado retardos porque todos

están trabajando con subtareas que pertenecen al camino crítico del sistema. El retardo total debido a la gestión del grafo de subtareas y la aplicación de las técnicas de precarga y reutilización es de 40 ciclos de reloj. Ya que para este tamaño la frecuencia de reloj es de 100 MHz, el retardo es sólo de 400 ns, que es despreciable, especialmente si tenemos en cuenta que en este caso nuestro gestor ha escondido las latencias de reconfiguración de tres subtareas y previene una costosa reconfiguración mediante la reutilización de una subtarea.

Tabla 5. Retardos introducidos por el gestor

Evento	Nuevo grafo	Fin de rec. (subtarea 1)	Fin de ej. (subtarea 1)	Fin de ej. (subtarea 3)
Ciclos	16	2	11	11

Tabla 6. Coste de implementación para un gestor con una tabla asociativa de ocho entradas y tres URs

Módulo	Número de slices	Slices (%)	Block RAMs	RAMs (%)
Unidad de control	21	0.2%	0	0%
FIFO rec.	8	0.2%	1	1%
UR	36	0.2%	1	1%
Cola de eventos	8	0.2%	1	1%
Tabla asociativa	331	2%	1	1%
Gestor	454	3%	6	4%

El coste de cada uno de estos módulos y el coste total del gestor de ejecución se muestran en la tabla 6. Como se puede ver para este pequeño gestor, sólo se necesita el 3% de los recursos de la FPGA. El módulo más costoso es la tabla asociativa porque ha sido diseñada para proporcionar el máximo rendimiento. Sin embargo, el coste es todavía bastante reducido para este tamaño. La tabla asociativa es también el módulo con mayor retardo, por lo tanto es de interés el estudio de su coste y el retardo para diferentes tamaños. Estos datos se presentan en la tabla 7. Como se puede ver, el coste de la tabla es lineal con respecto al número de entradas, y la frecuencia de reloj soportada por el sistema se reduce 15-20% cada vez que duplicamos el tamaño de la tabla. Sin embargo, se debe remarcar que las FPGAs actuales incluyen soporte para el uso de múltiples relojes, por lo tanto, la reducción de la frecuencia de reloj sólo reduce el rendimiento de nuestro gestor, pero no el rendimiento de las subtareas que son ejecutadas en las URs, ya que estas pueden utilizar su propio reloj.

Tabla 7. Coste y frecuencia de reloj para distintos tamaños de la tabla asociativa

Tamaño	8	16	32	64
Slices (%)	2	4	8	16
Block RAMs(%)	0	0	0	0
Frecuencia de reloj	100	85	67	56

ANÁLISIS COMPARATIVO SOFTWARE

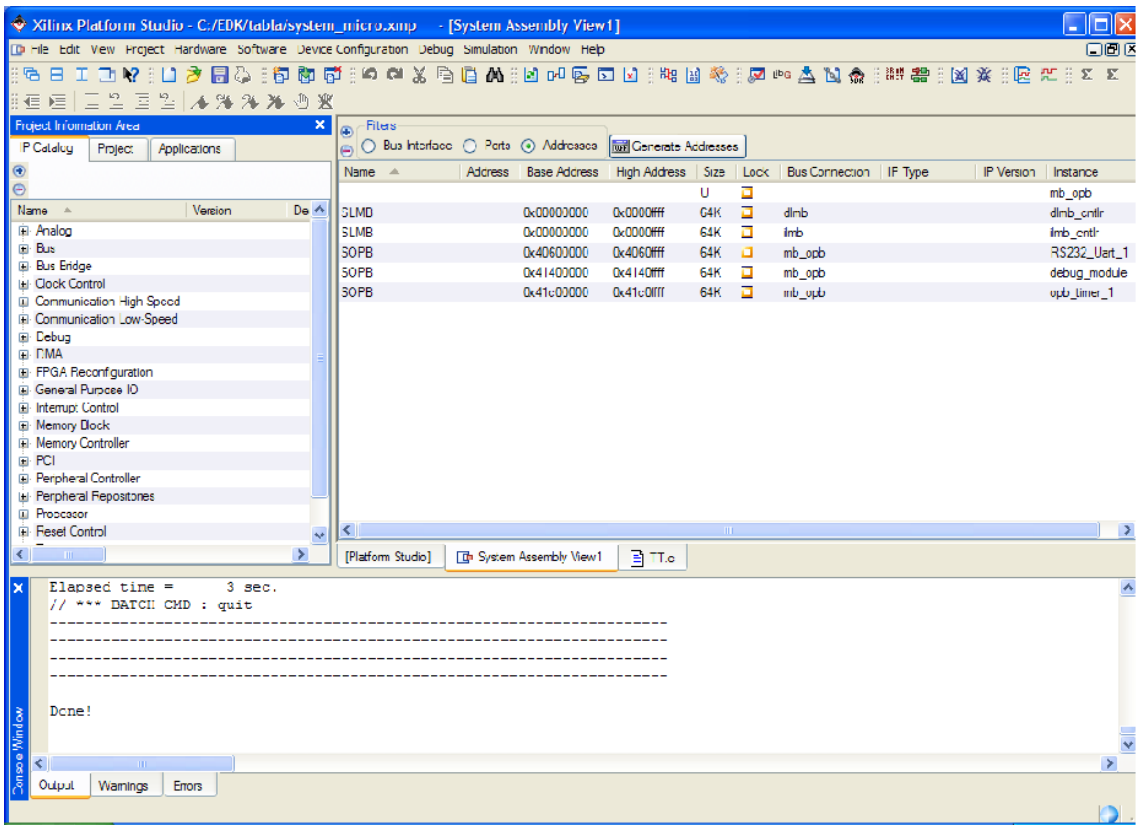
Introducción

Para mostrar la mejora de rendimiento de nuestro gestor HW respecto a una versión SW equivalente, se ha desarrollado un código en C equivalente a la tabla de tareas para ser ejecutado en un microprocesador *Microblaze.* El objetivo será comparar los tiempos obtenidos en ambas versiones y analizar el *speed-up* o mejora de rendimiento.

Para dicha comprobación habrá que tener en cuenta que en la versión SW no se consideren los tiempos de ejecución que se producen a causa de la gestión del controlador de la tabla de tareas. Por tanto, nos centraremos únicamente en los tiempos empleados en las operaciones de la tabla.

En la obtención de los tiempos, utilizaremos una FPGA *Virtex-II PRO*, el *Xilinx Platform Studio 8.2.02i* (EDK) y el respectivo código fuente en C (se incluye en el anexo X).

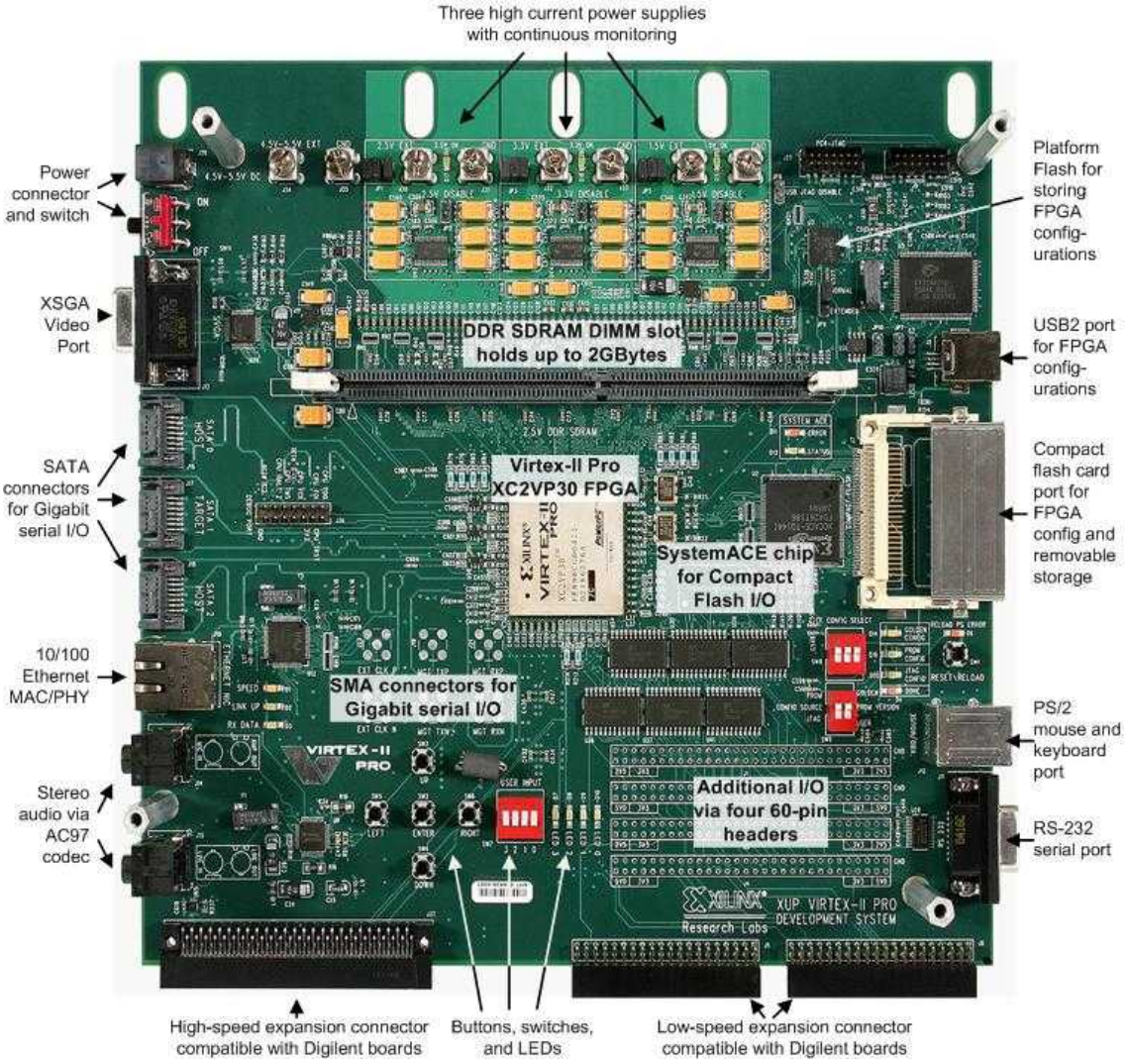
Figura 23. Ilustración del programa Xilinx Platform Studio 8.2.02i



La toma de tiempos en la FPGA *Virtex-II PRO* se ha realizado sobre un procesador *MicroBlaze*. Este procesador de 32-bit está basado en una arquitectura RISC con 32 registros de 32 bits (LUT RAM) y con separación de instrucciones para datos y accesos a memoria. Además, en la configuración realizada para la toma de tiempos, se utilizó un reloj a 100 MHz de frecuencia, sin cache y con 8Kb de memoria local.

Los tiempos se han obtenido con la ayuda de un temporizador *(timer)* situado en el bus OPB de dicho procesador. Para medir los tiempos empleados en las operaciones de la tabla, expresados en ciclos, se han ignorado aquellos correspondientes a los necesarios para las consultas al temporizador.

Figura 24. Diagrama explicativo de la FPGA Virtex-II PRO



Repaso de las operaciones de la tabla

En apartados anteriores ya se comentó detalladamente cuáles y en qué consistían las operaciones de la tabla del gestor de tareas; aun así, se hará un breve repaso que nos servirá para recalcar las características de la versión SW. Las operaciones de la tabla son las siguientes:

Añadir subtarea: añade la subtarea con sus datos en la tabla.
Subtarea preparada: devuelve la subtarea que está lista para ejecutarse.
Actualizar dependencias: elimina de la lista de predecesores de todas las subtareas el identificador de la subtarea que ha sido eliminada de la tabla.
Eliminar subtarea: se encarga de eliminar la subtarea de la tabla.

Para la comparación respecto a la versión SW se han de tener en consideración los tiempos que se han obtenido en la versión HW. En la tabla 8 se muestran dichos tiempos.

Tabla 8. Ciclos de las operaciones de la versión SW

Operación	Tiempo (ciclos)
Añadir subtarea	1
Tarea preparada	1
Actualizar dependencias	nº sucesores
Eliminar subtarea	1+ Actualizar dependencias

Comparación HW/SW teórica

En este apartado se realizará una comparativa teórica entre la tabla de tareas HW y la tabla de tareas software. En primer lugar, se obtendrán las complejidades a nivel de ciclo de las operaciones de la tabla HW, seguidamente las complejidades SW a nivel de instrucción y, posteriormente se procederá a su comparación. Se recuerda que la implementación HW y SW difieren entre ellas puesto que se ha orientado cada una a la búsqueda de la mayor eficiencia. Aunque ambas utilizan una lista de subtareas, la implementación SW está basada en una lista de predecesores mientras que la implementación HW está basada en una lista de sucesores. Además, indicar también que la tabla de tareas posee una mayor funcionalidad en el contexto HW que en el SW.

Las complejidades de las operaciones *Añadir subtarea* y *Subtarea preparada* de la tabla en la versión HW son constantes (1 ciclo). La operación *Actualizar dependencias* tarda tantos ciclos como sucesores posee la subtarea a eliminar, por tanto, su complejidad es O(p) siendo p la longitud de la lista de sucesores. La operación *Eliminar subtarea* utiliza 1 ciclo en la eliminación de la subtarea pero invoca *Actualizar dependencias* para actualizar el estado de la tabla de tareas, por tanto, su complejidad es O(p).

Las complejidades en la versión SW se han tomado a nivel de instrucción. Estas complejidades son una estimación inferior a la que se obtendrían a nivel de ciclo, ya que muchas operaciones implican dos o más ciclos además de las instrucciones de salto y control. Las operaciones *Añadir subtarea* y *Subtarea preparada* deben recorrer la lista para localizar la posición de inserción en el caso de *Añadir subtarea* y de extracción en el caso de *Subtarea preparada,* por tanto, las complejidades de estas dos operaciones son de O(n) siendo n el número de subtareas en la lista. Las complejidades de las otras dos operaciones, *Actualizar dependencias* y *Eliminar subtarea*, son dependientes (*Eliminar subtarea* actualiza posteriormente la tabla invocando a la operación *Actualizar dependencias*). El mayor orden de complejidad lo provoca la actualización de las dependencias que necesita recorrer la lista de subtareas completamente y eliminar en cada lista de predecesores el identificador de la subtarea eliminada (si es que ésta pertenece a esa lista), por tanto, la complejidad de *Actualizar dependencias* es O(n.m) siendo n el número de subtareas en la lista y m el número de elementos en la lista de predecesores. *Eliminar subtarea* aprovecha el recorrido de la lista para eliminar el nodo de la subtarea a eliminar, por tanto, la complejidad es la misma: O(n.m).

Como se indicó anteriormente, las complejidades han sido tomadas a diferente nivel. Para comparar los resultados teóricos se debe tener en cuenta que las complejidades a nivel de instrucción son inferiores en muchos casos a las tomadas a nivel de ciclo y pueden engañar al lector sobre la ganancia que se obtiene realmente. El motivo es que cada operación implica casi siempre más de un ciclo. Por ejemplo, el recorrido de la lista requiere por cada elemento dos comparaciones (la comparación del bucle, y la comparación del elemento en esa posición con el buscado), actualización del índice y la instrucción de salto, es decir, que por cada elemento se emplean mínimo 4 ciclos. Por tanto, aunque la complejidad puede seguir siendo O(n) y la complejidad de la versión HW O(1), realmente solo emplea 1 ciclo, obteniéndose una ganancia mínima en la versión HW de 4*(n/2)/1 (2n) en las operaciones *Añadir tarea* y *Subtarea preparada*. Para las operaciones *Actualizar dependencias* y *Eliminar subtarea* ocurre lo mismo. En este caso, por cada elemento, a parte de los 4 ciclos introducidos por el bucle de recorrido de la lista de subtareas hay que añadir otros 4 ciclos producidos por el recorrido de cada lista de predecesores. Por tanto, la ganancia mínima es (4(n/2)*4(m/2))/p. Si suponemos

que la representación con las listas de sucesores y las listas de predecesores es equivalente (m=p) la ganancia mínima obtenida es 4n. En un análisis general, para cada nodo a tratar en estos sistemas se realizan mínimo las operaciones de adicción, *Subtarea preparada* y borrado, asi que la ganancia por nodo se puede aproximar a 2n+2n+2n*2m/1+p. Teniendo en cuenta la misma supoción que antes (m=p) se obtiene 4n+4n*m/1+m. Despejando se obtiene 4n * ((1+m)/(1+m)). Para valores grandes de m (m → ∞), se puede aproximar a una ganancia mínima a 4n.

Tabla 9. Comparativa teórica

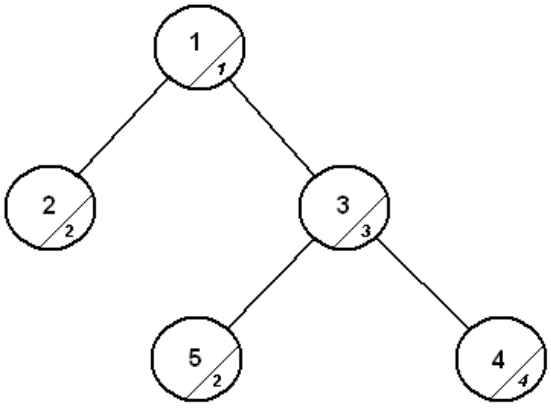
Operación	Complejidad HW (a nivel de ciclo)	Complejidad SW (a nivel de instrucción)
Añadir subtarea	O(1)	O(n)
Subtarea preparada	O(1)	O(n)
Actualizar dependencias	O(p)	O(n.m)
Eliminar subtarea	O(p)	O(n.m)

Nota: siendo p – longitud lista sucesores,
n – longitud lista de subtareas,
m – longitud lista predecesores.

Obtención tiempos versión SW con un ejemplo práctico

Una vez analizadas y comprendidas las operaciones de la tabla de tareas, se realizó la toma de tiempos. Todos los tiempos se obtuvieron a partir del grafo de ejemplo que mostramos en la figura 25. Los números pequeños a la derecha de cada nodo representan la UR en que debe ser ejecutada la tarea correspondiente a dicho nodo.

Figura 25. Grafo a analizar para la comparación de tiempos



Subtarea: 1
Tiempo de ejecución: 2.0
Tiempo de reconfiguración: 2.0
Lista predecesores:

Subtarea: 2
Tiempo de ejecución: 2.0
Tiempo de reconfiguración: 1.0
Lista predecesores: 1

Subtarea: 3
Tiempo de ejecución: 3.0
Tiempo de reconfiguración: 1.0
Lista predecesores: 1

Subtarea: 4
Tiempo de ejecución: 3.0
Tiempo de reconfiguración: 2.0
Lista predecesores: 3

Subtarea: 5
Tiempo de ejecución: 5.0
Tiempo de reconfiguración: 2.0
Lista predecesores: 2,3

El grafo mostrado se procesa y guarda en la memoria quedando estructurado en una lista de tareas cada una de ellas con la información relativa a ella: su identificador, su tiempo de ejecución, su tiempo de reconfiguración y una lista con los identificadores de las tareas

predecesoras a ella. Se obtendría una estructura como se muestra al lado derecho del grafo de la figura 25.

Como se indicó anteriormente, los tiempos se obtuvieron mediante el temporizador situado en el bus OPB también conocido como *opb_timer*. Las sentencias de código empleadas para la obtención de tiempos son las indicadas en la figura 26:

Figura 26. Código C para el uso del temporizador

```
//INICIALIZACIÓN:
XTmrCtr counter;
XStatus Status;
Xuint32 t1, t2;
Status = XTmrCtr_Initialize(&counter,XPAR_OPB_TIMER_1_DEVICE_ID);
XTmrCtr_Reset(&counter,0);
XTmrCtr_Start(&counter, 0);

//OBTENCIÓN TIEMPO (ciclos):
t1=XTmrCtr_GetValue(&counter, 0);
t2=XTmrCtr_GetValue(&counter, 0);

//PARADA:
XTmrCtr_Stop(&counter, 0);
```

Nota: XTmrCtr es una librería de Xillinx.

En la primera fase para la toma de tiempos, se inicializó la tabla y se incluyeron las subtareas para conseguir el grafo indicado en la figura 25.

Tabla 10. Ciclos de la operación "añadir subtarea"

Subtarea añadida	Ciclos empleados
Subtarea 1	43
Subtarea 2	56
Subtarea 3	56
Subtarea 4	56
Subtarea 5	65
TOTAL	276

En la siguiente fase, se analizó el tiempo que tarda en indicar si una subtarea está preparada para su ejecución, cuánto tiempo tarda en borrarse y cuánto tiempo tarda en actualizar sus dependencias.

Finalmente, a partir de los resultados obtenidos, se señalará el porqué de dichos tiempos (expresados en ciclos). Respecto a la inclusión de las subtareas en la tabla, se observa que cuantas más subtareas se hayan incluido en la lista antes de la subtarea a añadir, más ciclos se necesitan para insertarla. Asimismo, se tarda más tiempo en insertar una subtarea cuantos más predecesores tenga. Se obtiene un tiempo medio de inserción en este caso de 55 ciclos.

La operación *subtarea preparada* consume 41 ciclos en el ejemplo. Este tiempo es constante porque en este ejemplo es siempre la primera subtarea en la lista la que está disponible para ejecución. Por tanto, el tiempo medio es igual a 41 ciclos.

Respecto a la actualización de dependencias, se recuerda que se incluye en la operación eliminación de una subtarea. Se observa que la actualización de dependencias va ligada al número de subtareas que hay en la lista y al número de subtareas que dependen de ella (si dependen de ella, hay que eliminarla, lo que conlleva más ciclos). El tiempo medio de esta operación para el ejemplo son 165 ciclos.

		Subtarea1		Subtarea2		Subtarea3	
Subtarea preparada	41		41		41		41
Eliminar subtarea		898		681		558	
Actualizar dependencias		324		217		204	

Tabla 11. Ciclos que tardan las operaciones restantes

	Subtarea4		Subtarea5	**Total**
Subtarea preparada		41		205
Eliminar subtarea	300		157	2594
Actualizar dependencias	56		23	824

Para la operación de eliminación de una subtarea, a parte de la gestión de eliminación de dependencias conlleva la eliminación del nodo que representa la subtarea. El tiempo medio son aproximadamente 520 ciclos en este caso. De este tiempo, el 30% se emplea en la operación de actualización de dependencias.

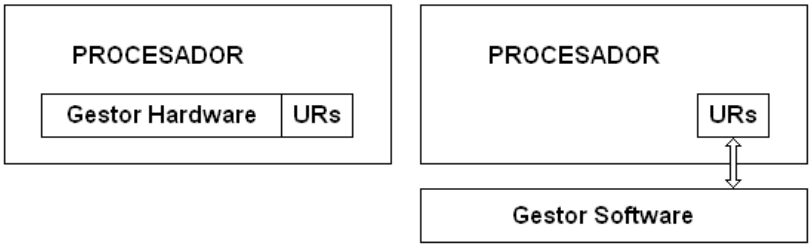
Si se quieren obtener estos tiempos en segundos, hay que recordar que el procesador software Microblaze tenía una frecuencia de reloj de 100 MHz. Se obtiene por tanto la equivalencia de 1 ciclo igual a 10 ns.

Comparación de tiempos (HW/SW)

1) *Comparación tiempos mediante el ejemplo práctico (HW/SW)*

Para la comparación de tiempos, no se ha tenido en cuenta el sobrecoste que implica la gestión de la tabla de tareas, así como el sobrecoste provocado por la comunicación entre el gestor software y las URs.

Figura 27. Comparativa entre los esquemas de diferentes versiones (HW/SW)



Si en la comparación se tiene en cuenta que el nº de sucesores medio del grafo es uno, se obtienen los resultados expresados en la tabla 12.

Tabla 12. Comparativa entre los ciclos que tardan las distintas operaciones

Operación	Tiempo HW (ciclos)	Tiempo SW (ciclos)	Ganacia (SW/HW)
Añadir subtarea	1	55	55
Subtarea preparada	1	41	41
Actualizar dependencias	2.5	156	62.4
Eliminar subtarea	1	364	364
TOTAL	5	616	123

Examinando y analizando la tabla 12 se observa que, sin considerar otros sobrecostes de la versión SW, se obtiene un sistema 123 veces más rápido.

2) *Comparación tiempos en* general *(HW/SW)*

Como ya se indicó anteriormente, a los tiempos obtenidos en la versión SW hay que añadir el sobrecoste de comunicación entre las URs y el procesador, además de la sobrecarga del controlador SW de la tabla de subtareas respecto a la versión HW del controlador (ganancia añadida), por tanto, estos resultados son inferiores a los mejores reales que el sistema puede proporcionar. Además, en el ejemplo se ha utilizado una lista de subtareas pequeña (5 subtareas). Como ya vimos en el caso teórico, la ganancia aumenta según crece el número de subtareas en la lista y la complejidad del grafo a la hora de tratar con las dependencias entre nodos.

CONCLUSIONES

En esta memoria hemos presentado un gestor de ejecución para un sistema HW implementado directamente en hardware. Su objetivo es mejorar la eficiencia de la gestión de tareas en sistemas multiprocesador reconfigurables, reduciendo las costosas comunicaciones HW/SW e incluyendo soporte HW para tratar operaciones con tipos de datos complejos rápidamente.

Este gestor recibe como entrada grafos de tareas planificados, y garantiza su correcta ejecución teniendo en cuenta las dependencias entre subtareas. Aparte, aplica dos técnicas de optimización para reducir el sobrecoste debido a las reconfiguraciones, llamadas reutilización de subtareas y precarga de reconfiguraciones. Con estas técnicas se ha eliminado en torno al 70% de la latencia de reconfiguración.

Los resultados experimentales demuestran que el coste de este módulo es muy reducido para sistemas pequeños/medianos y que proporciona un buen rendimiento. También demuestra que la tabla asociativa es el componente más crítico del sistema.

TRABAJO FUTURO

Podemos considerar que prácticamente todos los objetivos propuestos al inicio del desarrollo del proyecto han llegado a cumplirse: hemos diseñado, implementado y testeado un gestor para la ejecución de tareas HW, así como una mini-versión SW para comparar el módulo crítico del sistema (la tabla de tareas). De este modo hemos podido comprobar que es una buena elección optar por el HW dinámicamente reconfigurable para ejecutar tareas dadas como grafos de subtareas, y que las operaciones de gestión que se realizan en dichos grafos se ejecutan mucho más rápidamente en su versión HW que en su versión SW (tabla 12).

No obstante, debido quizá a falta de tiempo para continuar en su desarrollo, nos habría gustado implementar una versión SW del sistema completo para así poder realizar una comparativa a más alto nivel y más "*justa*". De este modo podríamos comparar cuánta mejora se obtiene al ejecutar una tarea completa en el sistema y no sólo en las operaciones de gestión de grafos en la tabla de tareas. Hablamos de *mejora* sin haber comprobado efectivamente que existe como tal; pero tenemos motivos para asegurar que una versión HW del sistema completo se comportará mejor que la versión SW del sistema completo, y probablemente el *speed-up* del mismo, sea mayor que el *speed-up* para la tabla de tareas. En primer lugar, las comunicaciones directas entre el HW implementado por nosotros y sintetizado en la FPGA son más rápidas que las que se establecerían entre un procesador y las URs del sistema en una versión completamente SW. En segundo lugar, sabemos que la ejecución HW tarda 137 ciclos para el grafo de ejemplo (figura 25), y solamente la tabla de tareas SW tarda más que todo el sistema HW (tabla 12); y cuantos más nodos tengan los grafos que se ejecuten, mejor se comporta la versión HW frente a la versión SW. No obstante, aunque se considere lo anteriormente mencionado, sería necesario ampliar nuestro estudio comparativo SW/HW para disponer de datos más fiables y realistas.

Es evidente que todo diseño HW se puede mejorar: se puede hacer que trabaje más rápido (a menor frecuencia de reloj), más eficiente (que realice las operaciones en menos ciclos) y que, al ser sintetizado en una FPGA, consuma menos recursos (en términos de área). Nuestro gestor no es una excepción. En concreto, queremos diseñar una tabla asociativa más escalable, y también reducir su retardo mediante la aplicación de técnicas de anticipación de operandos en la red iterativa y/o añadiendo algún ciclo extra. Esto haría que la frecuencia mínima de reloj a la que el sistema funcionase correctamente se redujera, con la consecuente ganancia en velocidad.

Como tercera ampliación, nos parece interesante intercambiar datos entre la FPGA y un PC, mediante algún protocolo de comunicaciones, como Ethernet o RS232. De este modo, se puede observar a través de un monitor la evolución del estado del sistema en tiempo de ejecución, sin tener que recurrir al uso de LEDs y displays de la FPGA. No obstante, esta ampliación requería de una amplia labor de investigación, debida a la falta de soporte que proporcionan los fabricantes en este sentido, y que se salía de los objetivos iniciales del proyecto, al proporcionar la simulación Post Place & Route la misma información.

Por último, y como trabajo a medio-largo plazo, se puede diseñar un planificador de tareas que interactúe con nuestro gestor. De este modo, tendríamos un sistema completo que recibiría el grafo de una tarea a ejecutar, ésta sería analizada en tiempo de diseño y/o tiempo de ejecución (según los algoritmos utilizados) y el gestor recibiría esta planificación para llevarla a cabo. En este sentido, hay trabajos relacionados en los que se plantea la posibilidad de crear un planificador de tareas que se ejecutan en distintas unidades reconfigurables utilizando técnicas de precarga. Asimismo, se plantea la posibilidad de aplicar técnicas de minimización de costes de reconfiguración en sistemas dinámicamente reconfigurables a una planificación dada, con lo que se podrían incorporar esas ideas. Todas estas ampliaciones ayudarían, sin duda, a proporcionar soporte a la investigación del HW reconfigurable en general y al desarrollo de una tecnología con posibles usos comerciales en el futuro; algo que nos parece especialmente interesante.

GLOSARIO DE TÉRMINOS

Asíncrono:
Hace referencia al suceso que no tiene lugar en total correspondencia temporal con otro suceso. Si se refiere a una comunicación asíncrona, ésta es en la que se realiza el envío de datos sin la sincronización de un reloj externo.

Bit:
Dígito en el sistema binario.

Bus:
Conjunto de conductores eléctricos en forma de pistas metálicas impresas sobre la placa base del computador, por donde circulan las señales que corresponden a los datos binarios del lenguaje máquina. Si el bus es de datos, este conectará dos dispositivos hardware.

Byte:
8 bits.

Chip:
Circuito integrado o pastilla en la que se encuentran todos o casi todos los componentes necesarios para que un ordenador pueda realizar alguna función.

Ciclo:
Un ciclo es la distancia temporal entre el principio y el final de una onda completa.

Circuito impreso:
Medio para sostener mecánicamente y conectar eléctricamente componentes electrónicos, a través de rutas o pistas de material conductor, grabados desde hojas de cobre laminadas sobre un sustrato no conductor.

Depuración:
Proceso de mejora de un sistema hasta que realiza su función correctamente.

Empotrado:
Hubicado dentro del chip.

Esquemático:
Es un diagrama, dibujo o boceto que detalla los elementos de un sistema.

Evento:
Acontecimiento ocurrido en el sistema.

FIFO:
First In, First Out. Protocolo que implementa una cola: lo que viene primero, se maneja primero, lo que viene segundo espera hasta que lo primero haya sido manejado, etc.

Flanco:
Transición del nivel bajo al alto (flanco de subida) o del nivel alto al bajo (flanco de bajada) de una señal.

FPGA:
Dispositivo donde se pueden programar infinidad de diseños digitales distintos.

Gestor:
Elemento que se encarga de organizar y dar órdenes a otros elementos de menor nivel en la jerarquía de diseño.

Grafo:
Conjunto de nodos relacionados mediante aristas de simple o doble sentido.

Hardware:

Es un término general usado para describir artefactos físicos de una tecnología.

Planificación:
Modelo de ejecución para una tarea.

Precarga:
Técnia que preve y lleva a cabo una reconfiguración parcial con anterioridad a su utilización.

Predecesor:
Que se encuentra con anterioridad en la jerarquía.

Prototipo:
Ejemplar original o primer molde en que se fabrica un diseño.

Reconfiguración:
Proceso en el que parte de la FPGA (reconfiguración parcial) o toda (reconfiguración total) se reprograma.

Síncrono:
Hace referencia al suceso que tiene lugar en correspondencia temporal con otro suceso. Si se refiere a una comunicación síncrona, ésta es en la que se realiza el envío de datos bajo la sincronización de un reloj externo.

Speed-up:
Cociente que nos dice la mejora de rendimiento obtenida.

Subtarea:
Cada una de las partes de una tarea con significado propio.

Sucesor:
Que se encuentra con posterioridad en la jerarquía.

Unidad reconfigurable:
Parte de la FPGA que puede ser reconfigurada de modo independiente.

VHDL:
Very High Speed Integrate Circuit Hardware Description Language, es un lenguaje de descripción y modelado, diseñado para describir la funcionalidad y la organización de sistemas *hardware* digitales, placas de circuitos, y componentes.

BIBLIOGRAFÍA

[1] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, R. Lauwereins, “Interconnection Network enable Fine-Grain Dynamic Multi-Tasking on FPGAs”, Proc. Of FPL’02, pp. 795-805, 2002.

[2] J. Noguera, M. Badia., “Power-Performance Trade-Offs for Reconfigurable Computing”. CODES+ISSS.pp. 116-121. 2004

[3] K. N. Vikram, V. Vasudevan. “Mapping Data-Parallel Tasks Onto Partially Reconfigurable Hybrid Processor Architectures”. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Volume 14, Issue 9, Sept. 2006, pp.1010-1023.

[4] H. Walder, M. Platzner, “Online Scheduling for Block-partitioned Reconfigurable Devices”. Proceedings of the Design Automation & Test in Europe Conference (DATE’03), pag. 10290-10295. 2003.

[5] J. Resano, D. Mozos, D Verkest, F Catthoor, "*A Reconfiguration Manager for Dynamically Reconfigurable Hardware*", IEEE Design&Test, Vol. 22, Issue 5, pp. 452-460. 2005.

[6] W. Fu, K. Compton. “An execution environment for reconfigurable computing”. Proc. of Field-Programmable Custom Computing Machines (FCCM), 2005, pp.149-158.

[7] J. Javier Resano, M. Elena Pérez, Daniel Mozos, Hortensia Mecha, Julio Septién. “Analyzing Communication Overheads during Hardware/Software Partitioning”. Elsevier Microelectronic Journal, vol. 34-11, pag. 1001-1007. 2003.

[8] R. P. Dick, G. Lakshminarayana, A. Raghunathan, and N.K. Jha. “Power analysis of embedded operating systems”. In Proceeding Design Automation Conference (DAC),2000.

[9] B.E. Saglam, V. Andmooney. “System-on-a-chip processor synchronization support in hardware”. In Proceedings of Design Automation and Test in Europe (DATE’01). 2001.

[10] V. Nollet, T. Marescaux, D. Verkest, J-Y. Mignolet, S. Vernalde. “Operating System controlled Networkon-Chip”. Proceedings of the Design Automation Conference (DAC), pag. 256-259, 2004.

[11] J. Noguera, M. Badia., “Multitasking on Reconfigurable Architectures: Microarchitecture Support and Dynamic Scheduling”. ACM Transactions on Embedded Computing Systems, Vol. 3, No. 2, pp 385-406. 2004

[12] Xilinx Inc., Virtex-II Pro and Virtex-II Pro X Platform FPGAs:Complete Data Sheet, Xilinx, San Jose, Calif, USA, 2005.

[13] Z. Li, S. Hauck, “Configuration prefetching techniques for partial reconfigurable coprocessor with relocation and defragmentation”, FPGA’02, pp. 187-195. 2002

[14] J. Resano, D. Mozos, F Catthoor, "A Hybrid Prefetch Scheduling Heuristic to Minimize at Runtime the Reconfiguration Overhead of Dynamically Reconfigurable HW" DATE05, pp.106-111. 2005

APÉNDICES

FUENTES VHDL DEL PROYECTO

Módulos Básicos

Biestable.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity biestable is
   port( clk, rst, ld : in std_logic;
      din : in std_logic;
      dout : out std_logic);
end biestable;

architecture biestableArch of biestable is
   signal cs, ns : std_logic;
   begin

   state:
   process( clk, rst )
      begin
         if (rst='1') then
            cs <= '1';
         elsif (clk'event and clk='1') then
            cs <= ns;
         end if;
   end process;

   next_state:
      ns <= din when ld='1' else cs;

   moore_output:
      dout <= cs;

end biestableArch;
```

Codificador.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity priorityEncoder is
    generic( n : integer := 3 );
    port(
       x : in std_logic_vector( 2**n-1 downto 0 );
       y : out std_logic_vector ( n-1 downto 0 );
       gs : out std_logic );
end priorityEncoder;

architecture priorityEncoderArch of priorityEncoder is
begin
    process ( x )
    begin
       y <= (others=>'0'); gs <= '0';
       for i in x'reverse_range loop
        if x(i)='1' then
            y <= conv_std_logic_vector( i, n );
            gs <= '1';
        end if;
       end loop;
    end process;
```

```
end priorityEncoderArch;
```

Contador.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity contador is
   generic( n : integer := 8 );
   port( clk, rst, ld, dec : in std_logic;
      din : in std_logic_vector( n-1 downto 0 );
      dout : out std_logic_vector( n-1 downto 0 ) );
end contador;

architecture contadorArch of contador is
   signal cs, ns : std_logic_vector( n-1 downto 0 );
   begin

   state:
   process( clk, rst )
     begin
        if rst='1' then cs <= (others=>'0');
        elsif clk'event and clk='1' then cs <= ns;
        end if;
   end process;

   next_state:
   process( cs, ld, dec, din )
      begin
         if ld='1' then ns <= din;
         elsif dec='1' then ns <= cs - 1;
                  -- no es necesario detectar el final de cuenta
         else ns <= cs;
         end if;
   end process;

   moore_output: dout <= cs;

end contadorArch;
```

ContadorInc.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity contadorInc is
   generic( n : integer := 8 );
   port( clk, rst, inc : in std_logic;
      dout : out std_logic_vector( n-1 downto 0 ) );
end contadorInc;

architecture contadorIncArch of contadorInc is
   signal cs, ns : std_logic_vector( n-1 downto 0 );
   begin

   state:
   process( clk, rst )
     begin
        if rst='1' then cs <= (others=>'0');
        elsif clk'event and clk='1' then cs <= ns;
        end if;
   end process;

   next_state:
   process( cs, inc )
```

```
      begin
         if inc='1' then ns <= cs + 1;
         else ns <= cs;
         end if;
   end process;

   moore_output: dout <= cs;

end contadorIncArch;
```

ContadorIncDec.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity contadorIncDec is
   generic( n : integer := 8 );
   port( clk, rst, inc, dec : in std_logic;
      dout : out std_logic_vector( n-1 downto 0 ) );
end contadorIncDec;

architecture contadorIncDecArch of contadorIncDec is
   signal cs, ns : std_logic_vector( n-1 downto 0 );
   begin

   state:
   process( clk, rst )
     begin
        if rst='1' then cs <= (others=>'0');
        elsif clk'event and clk='1' then cs <= ns;
        end if;
   end process;

   next_state:
   process( cs, dec, inc )
      begin
         if inc='1' then ns <= cs + 1; -- prioridad al incremento
         elsif dec='1' then ns <= cs - 1;
         else ns <= cs;
         end if;
   end process;

   moore_output: dout <= cs;

end contadorIncDecArch;
```

ContadorIncDecMismoCiclo.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity contadorIncDecMismoCiclo is
   generic( n : integer := 8 );
   port( clk, rst, inc, dec : in std_logic;
      dout : out std_logic_vector( n-1 downto 0 ) );
end contadorIncDecMismoCiclo;

architecture contadorIncDecMismoCicloArch of contadorIncDecMismoCiclo is
   signal cs, ns : std_logic_vector( n-1 downto 0 );
   begin

   state:
   process( clk, rst )
```

```
      begin
         if rst='1' then cs <= (others=>'0');
         elsif clk'event and clk='1' then cs <= ns;
         end if;
   end process;

   next_state:
   process( cs, dec, inc )
      begin
         if inc='1' and dec='1' then
            ns <= cs;
         elsif inc='1' then
      ns <= cs + 1;
         elsif dec='1' then
      ns <= cs - 1;
         else
      ns <= cs;
         end if;
   end process;

   moore_output: dout <= cs;

end contadorIncDecMismoCicloArch;
```

Decodificador.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity decoder is
    generic( n : integer := 3 );
    port(
       x : in std_logic_vector( n-1 downto 0 );
       en : in std_logic;
       y : out std_logic_vector ( 2**n-1 downto 0 ) );
end decoder;

architecture decoderArch of decoder is
begin
    process ( x, en )
    begin
       y <= (others=>'0');
       if en='1' then
          y(conv_integer(unsigned(x))) <= '1';
       end if;
    end process;
end decoderArch;
```

Registro.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity registro is
   generic( n : integer := 8);
   port( clk, rst, ld : in std_logic;
      din : in std_logic_vector( n-1 downto 0 );
      dout : out std_logic_vector( n-1 downto 0 ) );
end registro;

architecture registroArch of registro is
   signal cs, ns : std_logic_vector( n-1 downto 0 );
   begin

   state:
```

```
    process( clk, rst )
       begin
          if (rst='1') then
             cs <= (others=>'0');
          elsif (clk'event and clk='1') then
             cs <= ns;
          end if;
    end process;

    next_state:
       ns <= din when ld='1' else cs;

    moore_output:
       dout <= cs;

end registroArch;
```

FIFOs

Fifo3Bits.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;


entity fifo3bits is
   generic (numentradaslog : integer := 3 );
   port (entradadatos: in std_logic_vector(2 downto 0);
     clk, rst, leer, escribir, enable : in std_logic;
          salidadatos: out std_logic_vector(2 downto 0);
     fifollena, fifovacia: out std_logic);
end fifo3bits;

architecture fifo3bitsarch of fifo3bits is

component ramb16_s4
-- AAAsynthesis translate_off
generic (
       srval : bit_vector := x"0";
       write_mode : string := "no_change");
   -- AAAsynthesis translate_on
   port (do : out std_logic_vector (3 downto 0);
         addr : in std_logic_vector (11 downto 0);
         clk : in std_ulogic;
         di : in std_logic_vector (3 downto 0);
         en : in std_ulogic;
         ssr : in std_ulogic;
         we : in std_ulogic);
end component;

   component contadorinc
      generic( n : integer := 9 );
      port( clk, rst, inc : in std_logic;
            dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

  component contadorincdecmismociclo
      generic( n : integer := 8 );
      port( clk, rst, inc, dec : in std_logic;
         dout : out std_logic_vector( n-1 downto 0 ) );
   end component;
```

```
    signal salidacontadorprimero,
      salidacontadorultimo :std_logic_vector(numentradaslog-1 downto 0);
    signal salidacontadornumocupadas,
      ceros, unos :std_logic_vector(numentradaslog downto 0);
    signal incprimero, incultimo, escribirleer, enablefifo :std_logic;
    signal direccion: std_logic_vector(11 downto 0);
    signal entradafifo, salidafifo : std_logic_vector(3 downto 0);

begin

ram : ramb16_s4
    -- AAAsynthesis translate_off
   -- AAAsynopsys translate_on
port map (do => salidafifo,
             addr => direccion,
             clk => clk,
             di => entradafifo,
             en => enablefifo,
             ssr => rst,
             we => escribirleer);

    enablefifo <= leer or escribir;
    escribirleer <= '1' when (escribir='1') else '0';
    direccion(11 downto numentradaslog) <= (others=> '0');
    direccion (numentradaslog-1 downto 0) <=
           salidacontadorultimo when (escribir='1') else
           salidacontadorprimero;

    entradafifo (3) <= '0';
    entradafifo (2 downto 0) <= entradadatos;
    salidadatos <= salidafifo (2 downto 0);

    ceros <= (others =>'0');
    unos <= conv_std_logic_vector(2**numentradaslog,numentradaslog+1);

    contadorprimero: contadorinc generic map(numentradaslog)
           port map(clk, rst, incprimero, salidacontadorprimero);
    contadorultimo: contadorinc generic map(numentradaslog)
           port map(clk, rst, incultimo, salidacontadorultimo);

    incprimero <= '1' when (escribirleer ='0'  and enablefifo ='1') else '0';
    incultimo <= '1' when (escribirleer ='1'  and enablefifo ='1') else '0';

    contadornumocupadas: contadorincdecmismociclo
           generic map(numentradaslog+1)
           port map(clk, rst, incultimo, incprimero,
                                          salidacontadornumocupadas);

    fifollena <= '1' when (salidacontadornumocupadas = unos) else '0';
    fifovacia <= '1' when (salidacontadornumocupadas = ceros) else '0';

end fifo3bitsarch;
```

Fifo5Bits.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fifo5bits is
   generic (numentradaslog : integer := 3 );
   port (entradadatos: in std_logic_vector(4 downto 0);
     clk, rst, leer, escribir, enable : in std_logic;
     salidadatos: out std_logic_vector(4 downto 0);
     fifollena, fifovacia: out std_logic);
end fifo5bits;
```

```
architecture fifo5bitsarch of fifo5bits is

component ramb16_s9
-- AAAsynthesis translate_off
generic (
       srval : bit_vector := x"0";
       write_mode : string := "no_change"
);
-- AAAsynthesis translate_on
   port (do : out std_logic_vector (7 downto 0);
         dop : out std_logic_vector (0 downto 0);
         addr : in std_logic_vector (10 downto 0);
         clk : in std_ulogic;
         di : in std_logic_vector (7 downto 0);
         dip : in std_logic_vector (0 downto 0);
         en : in std_ulogic;
         ssr : in std_ulogic;
         we : in std_ulogic);
end component;

   component contadorinc
      generic( n : integer := 9 );
      port( clk, rst, inc : in std_logic;
            dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

   component contadorincdecmismociclo
      generic( n : integer := 8 );
      port( clk, rst, inc, dec : in std_logic;
               dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

   signal salidacontadorprimero,
        salidacontadorultimo :std_logic_vector(numentradaslog-1 downto 0);
   signal salidacontadornumocupadas,
        ceros, unos :std_logic_vector(numentradaslog downto 0);
   signal incprimero, incultimo, escribirleer, enablefifo :std_logic;
   signal direccion: std_logic_vector(10 downto 0);
   signal entradafifo, salidafifo : std_logic_vector(7 downto 0);
   signal dip : std_logic_vector (0 downto 0);

begin

ram : ramb16_s9
    -- AAAsynthesis translate_off
   -- AAAsynopsys translate_on
port map (do => salidafifo,
                dop => open,
                addr => direccion,
                clk => clk,
                di => entradafifo,
                dip => dip,
                en => enablefifo,
                ssr => rst,
                we => escribirleer);

    dip <= (others=> '0');

    enablefifo <= leer or escribir;
    escribirleer <= '1' when (escribir='1') else '0';
    direccion(10 downto numentradaslog) <= (others=> '0');
    direccion (numentradaslog-1 downto 0) <=
       salidacontadorultimo when (escribir='1') else salidacontadorprimero;

    entradafifo (7 downto 5) <= (others=>'0');
    entradafifo (4 downto 0) <= entradadatos;
    salidadatos <= salidafifo (4 downto 0);

    ceros <= (others =>'0');
    unos <= conv_std_logic_vector(2**numentradaslog,numentradaslog+1);
```

```
    contadorprimero: contadorinc generic map(numentradaslog)
           port map(clk, rst, incprimero, salidacontadorprimero);
    contadorultimo: contadorinc generic map(numentradaslog)
           port map(clk, rst, incultimo, salidacontadorultimo);

    incprimero <= '1' when (escribirleer ='0'  and enablefifo ='1') else '0';
    incultimo <= '1' when (escribirleer ='1'  and enablefifo ='1') else '0';

    contadornumocupadas: contadorincdecmismociclo
           generic map(numentradaslog+1)
           port map(clk, rst, incultimo, incprimero,
                                          salidacontadornumocupadas);

    fifollena <= '1' when (salidacontadornumocupadas = unos) else '0';
    fifovacia <= '1' when (salidacontadornumocupadas = ceros) else '0';

end fifo5bitsarch;
```

Fifo16Bits.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fifo16bits is
   generic (numentradaslog : integer := 3 );
   port (entradadatos: in std_logic_vector(15 downto 0);
     clk, rst, leer, escribir, enable : in std_logic;
     salidadatos: out std_logic_vector(15 downto 0);
     fifollena, fifovacia: out std_logic);
end fifo16bits;

architecture fifo16bitsarch of fifo16bits is

component ramb16_s18
-- aaasynthesis translate_off
generic (
       srval : bit_vector := x"0";
       write_mode : string := "no_change"
);
-- AAAsynthesis translate_on
   port (do : out std_logic_vector (15 downto 0);
         dop : out std_logic_vector (1 downto 0);
         addr : in std_logic_vector (9 downto 0);
         clk : in std_ulogic;
         di : in std_logic_vector (15 downto 0);
         dip : in std_logic_vector (1 downto 0);
         en : in std_ulogic;
         ssr : in std_ulogic;
         we : in std_ulogic);
end component;

   component contadorinc
      generic( n : integer := 9 );
      port( clk, rst, inc : in std_logic;
            dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

   component contadorincdecmismociclo
      generic( n : integer := 8 );
      port( clk, rst, inc, dec : in std_logic;
         dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

   signal salidacontadorprimero,
      salidacontadorultimo :std_logic_vector(numentradaslog-1 downto 0);
```

```
    signal salidacontadornumocupadas,
      ceros, unos :std_logic_vector(numentradaslog downto 0);
    signal incprimero, incultimo, escribirleer, enablefifo :std_logic;
    signal direccion: std_logic_vector(9 downto 0);
    signal entradafifo, salidafifo : std_logic_vector(15 downto 0);
    signal dip: std_logic_vector(1 downto 0);

begin

ramb16_s36_instance_name5 : ramb16_s18
port map (do => salidafifo,
             dop => open,
             addr => direccion,
             clk => clk,
             di => entradafifo,
             dip => dip,
             en => enablefifo,
             ssr => rst,
             we => escribirleer);

    dip <= (others=>'0');

    enablefifo <= leer or escribir;
    escribirleer <= '1' when (escribir='1') else '0';
    direccion(9 downto numentradaslog) <= (others=> '0');
    direccion (numentradaslog-1 downto 0) <=
      salidacontadorultimo when (escribir='1') else salidacontadorprimero;

    entradafifo (15 downto 0) <= entradadatos;
    salidadatos <= salidafifo;

    ceros <= (others =>'0');
    unos <= conv_std_logic_vector(2**numentradaslog,numentradaslog+1);

    contadorprimero: contadorinc generic map(numentradaslog)
         port map(clk, rst, incprimero, salidacontadorprimero);
    contadorultimo: contadorinc generic map(numentradaslog)
         port map(clk, rst, incultimo, salidacontadorultimo);

    incprimero <= '1' when (escribirleer ='0'  and enablefifo ='1') else '0';
    incultimo <= '1' when (escribirleer ='1'  and enablefifo ='1') else '0';

    contadornumocupadas: contadorincdecmismociclo
         generic map(numentradaslog+1)
         port map(clk, rst, incultimo, incprimero, salidacontadornumocupadas);

    fifollena <= '1' when (salidacontadornumocupadas = unos) else '0';
    fifovacia <= '1' when (salidacontadornumocupadas = ceros) else '0';

end fifo16bitsarch;
```

Fifo30Bits.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fifo30bits is
    generic (numentradaslog : integer := 3 );
    port (entradadatos: in std_logic_vector(29 downto 0);
      clk, rst, leer, escribir, enable : in std_logic;
      salidadatos: out std_logic_vector(29 downto 0);
      fifollena, fifovacia: out std_logic);
end fifo30bits;

architecture fifo30bitsarch of fifo30bits is
```

```
component ramb16_s36
-- AAAsynthesis translate_off
generic (
       srval : bit_vector := x"0";
       write_mode : string := "no_change"
);
-- AAAsynthesis translate_on
   port (do : out std_logic_vector (31 downto 0);
         dop : out std_logic_vector (3 downto 0);
         addr : in std_logic_vector (8 downto 0);
         clk : in std_ulogic;
         di : in std_logic_vector (31 downto 0);
         dip : in std_logic_vector (3 downto 0);
         en : in std_ulogic;
         ssr : in std_ulogic;
         we : in std_ulogic);
end component;

   component contadorinc
      generic( n : integer := 9 );
      port( clk, rst, inc : in std_logic;
            dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

   component contadorincdecmismociclo
      generic( n : integer := 8 );
      port( clk, rst, inc, dec : in std_logic;
         dout : out std_logic_vector( n-1 downto 0 ) );
   end component;

   signal salidacontadorprimero,
      salidacontadorultimo :std_logic_vector(numentradaslog-1 downto 0);
   signal salidacontadornumocupadas,
      ceros, unos :std_logic_vector(numentradaslog downto 0);
   signal incprimero, incultimo, escribirleer, enablefifo :std_logic;
   signal direccion: std_logic_vector(8 downto 0);
   signal entradafifo, salidafifo : std_logic_vector(31 downto 0);
   signal dip: std_logic_vector(3 downto 0);

begin

ramb16_s36_instance_name7 : ramb16_s36
port map (do => salidafifo,
             dop => open,
             addr => direccion,
             clk => clk,
             di => entradafifo,
             dip => dip,
             en => enablefifo,
             ssr => rst,
             we => escribir);

    dip <= (others=>'0');

    enablefifo <= leer or escribir;
    escribirleer <= '1' when (escribir='1') else '0';
    direccion(8 downto numentradaslog) <= (others=> '0');
    direccion (numentradaslog-1 downto 0) <=
     salidacontadorultimo when (escribir='1') else salidacontadorprimero;


    entradafifo (31 downto 30) <= (others=>'0');
    entradafifo (29 downto 0) <= entradadatos;
    salidadatos <= salidafifo (29 downto 0);

    ceros <= (others =>'0');
    unos <= conv_std_logic_vector(2**numentradaslog,numentradaslog+1);

    contadorprimero: contadorinc generic map(numentradaslog)
       port map(clk, rst, incprimero, salidacontadorprimero);
```

```
    contadorultimo: contadorinc generic map(numentradaslog)
       port map(clk, rst, incultimo, salidacontadorultimo);

    incprimero <= '1' when (escribirleer ='0'  and enablefifo ='1') else '0';
    incultimo <= '1' when (escribirleer ='1'  and enablefifo ='1') else '0';

    contadornumocupadas: contadorincdecmismociclo
        generic map(numentradaslog+1)
        port map(clk, rst, incultimo, incprimero, salidacontadornumocupadas);

    fifollena <= '1' when (salidacontadornumocupadas = unos) else '0';
    fifovacia <= '1' when (salidacontadornumocupadas = ceros) else '0';

end fifo30bitsarch;
```

Módulos compuestos

UR.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity ur is
--nº de bits para decir cuánto tiempo tarda en ejecutar y cargar. suponemos
que es lo mismo
    generic (numbitsetiqueta:natural:=3;
                  longitud:natural:=16;
                  numentradaslog:natural:=3;
                  long_eje_carga:natural:=5);
    port ( clk: in std_logic;
           rst: in std_logic;
           enable: in std_logic;
           entrada_fifo:in std_logic_vector(longitud-1 downto 0);
           escribir: in std_logic;
           comenzar_carga : in std_logic;
           comenzar_ejecucion : in std_logic;
           pedir_carga : out std_logic;
           pedir_ejecucion : out std_logic;
           sub_actual : out std_logic_vector(longitud-1 downto 0);
           tabla_llena: out std_logic;
           estado: out std_logic_vector(2 downto 0);
           vacia: out std_logic;
           escribirenfifoeventos: out std_logic;
           evento: out std_logic_vector(numbitsetiqueta+2-1 downto 0);
           concedidaescritura: in std_logic );
end ur;

architecture urarch of ur is


    component contador
       generic( n : integer := 5);
       port( clk, rst, ld, dec : in std_logic;
             din : in std_logic_vector( n-1 downto 0 );
             dout : out std_logic_vector( n-1 downto 0 ) );
    end component;

    component registro
       generic( n : integer := 8);
       port( clk, rst, ld : in std_logic;
             din : in std_logic_vector( n-1 downto 0 );
             dout : out std_logic_vector( n-1 downto 0 ) );
    end component;
```

```
    component fifo16bits
       generic (numentradaslog : integer := 3 )
       port (entradadatos: in std_logic_vector(15 downto 0);
         clk, rst, leer, escribir, enable : in std_logic;
         salidadatos: out std_logic_vector(15 downto 0);
         fifollena, fifovacia: out std_logic);
    end component;

    type states is (lee_fifo, finalizada, cargando_conf, esperando_ejecucion,
           ejecutando, escribirfinejecucion, escribirfinconfiguracion);

    signal currentstate, nextstate: states;

    signal salidafifo: std_logic_vector(longitud-1 downto 0);
    signal salidareg: std_logic_vector(numbitsetiqueta-1 downto 0);
    signal iguales, avanzar, ld_registro, tabla_vacia: std_logic;
    signal ceros, uno: std_logic_vector(long_eje_carga-1 downto 0);
    signal cargada, ejecutada: std_logic;
    signal dec_carga, dec_eje: std_logic;
    signal salida_cont_carga,
           salida_cont_eje: std_logic_vector(long_eje_carga-1 downto 0);

begin

    ceros <= (others =>'0');
    uno <= ceros + 1;

    fifo1: fifo13bits generic map(numentradaslog)
           port map(entrada_fifo, clk, rst, avanzar, escribir, enable,
                           salidafifo, tabla_llena, tabla_vacia);

    registrofifo: registro generic map(numbitsetiqueta)
           port map(clk, rst, ld_registro, salidafifo(12 downto 10),
                           salidareg);

    cont_eje: contador generic map(long_eje_carga)
           port map(clk, rst, comenzar_ejecucion, dec_eje,
                salidafifo(2*long_eje_carga-1 downto long_eje_carga) ,
                salida_cont_eje);

    cont_carga: contador generic map(long_eje_carga)
           port map(clk, rst, comenzar_carga, dec_carga,
                salidafifo(long_eje_carga-1 downto 0) , salida_cont_carga);


    cargada <= '1' when (salida_cont_carga = uno ) else '0';
    ejecutada <= '1'    when (salida_cont_eje = uno ) else '0';

    dec_carga <= not cargada;
    dec_eje <= not ejecutada;

    iguales <= '1' when (salidareg = salidafifo(12 downto 10)) else '0';
    sub_actual <= salidafifo;
    vacia <= tabla_vacia;

     --control de la tabla

    stategen:
    process (escribir, iguales, tabla_vacia, currentstate, comenzar_carga,
                  cargada, comenzar_ejecucion, ejecutada, concedidaescritura)
    begin
      nextstate <= currentstate;
      case currentstate is

       when lee_fifo =>
                if (tabla_vacia='0' and escribir='0') then
                    nextstate<=finalizada;
                end if;
```

```
            when finalizada =>
                if (iguales='0' and comenzar_carga='1') then
                    nextstate<=cargando_conf;
                elsif (iguales='1') then
                    nextstate<=escribirfinconfiguracion;
                end if;

            when cargando_conf =>
                if (cargada='1') then
                    nextstate<=escribirfinconfiguracion;
                end if;

            when escribirfinconfiguracion =>
                if (concedidaescritura='1') then
                    nextstate<=esperando_ejecucion;
                end if;

            when esperando_ejecucion =>
                if (comenzar_ejecucion='1') then
                    nextstate<=ejecutando;
                end if;

            when ejecutando =>
                if (ejecutada='1') then
                    nextstate<=escribirfinejecucion;
                end if;

            when escribirfinejecucion =>
                if (concedidaescritura='1') then
                    nextstate<=lee_fifo;
                end if;

        end case;
    end process stategen;

    state:
    process (rst, clk)
    begin
      if (rst = '1') then
        currentstate <= lee_fifo;
      elsif (clk'event and clk='1') then
        currentstate <= nextstate;
      end if;
    end process state;

    evento(numbitsetiqueta+2-1 downto 2)<=
                  salidafifo(longitud-1 downto longitud-numbitsetiqueta);

    mooremealygen:
    process (escribir, iguales, tabla_vacia, currentstate, comenzar_carga,
     cargada, comenzar_ejecucion, ejecutada, concedidaescritura, salidafifo)
    begin
      case currentstate is

     when lee_fifo =>
                avanzar <= tabla_vacia nor escribir;
                pedir_carga<='0';
                pedir_ejecucion<='0';
                ld_registro<='0';
                estado<="000";
                escribirenfifoeventos <= '0';
                evento(1 downto 0)<="00";

            when finalizada =>
                pedir_carga <= not iguales;
                pedir_ejecucion<='0';
                avanzar<='0';
                ld_registro<='0';
                estado<="001";
                escribirenfifoeventos <= '0';
```

```
                evento(1 downto 0) <= "00";

            when cargando_conf =>
                pedir_ejecucion<='0';
                pedir_carga<='0';
                avanzar<='0';
                ld_registro<='0';
                estado<="010";
                escribirenfifoeventos <= '0';
                evento(1 downto 0) <= "00";

            when esperando_ejecucion =>
                pedir_ejecucion<='1';
                pedir_carga<='0';
                avanzar<='0';
                ld_registro<='1'; --copio la salida de la fifo en el registro
                estado<="100";
                escribirenfifoeventos <= '0';
                evento(1 downto 0) <= "00";

            when ejecutando =>
                pedir_ejecucion<='0';
                pedir_carga<='0';
                avanzar<='0';
                ld_registro<='0';
                estado<="101";
                escribirenfifoeventos <= '0';
                evento(1 downto 0) <= "00";

            when escribirfinejecucion =>
                pedir_ejecucion<='0';
                pedir_carga<='0';
                avanzar<='0';
                ld_registro<='0';
                estado<="110";
                escribirenfifoeventos <= '1';
                evento(1 downto 0) <= "10";

            when escribirfinconfiguracion =>
                pedir_ejecucion<='0';
                pedir_carga<='0';
                avanzar<='0';
                ld_registro<='0';
                estado<="011";
                escribirenfifoeventos <= '1';
                evento(1 downto 0) <= "01";

        end case;
    end process mooremealygen;

end urarch;
```

IterativaBásica.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity iterativaBasica is
   port( clk,rst,libre_ant, borrar, acierto_i, load_tarea : in std_logic;
         load_tarea_i,libre_sig : out std_logic);
end iterativaBasica;

architecture iterativaBasicaArch of iterativaBasica is

component biestable is
   port( clk, rst, ld : in std_logic;
```

```
      din : in std_logic;
      dout : out std_logic);
end component;

    signal salida_libre,salida_puertaAND3: std_logic;
    signal entradaBiestable, cargaBiestable: std_logic;

begin

  librei: biestable port map
                (clk,rst,cargaBiestable,entradaBiestable,salida_libre);


    entradaBiestable <= borrar and acierto_i;
    cargaBiestable <= salida_puertaAND3 or (borrar and acierto_i);
    salida_puertaAND3 <= (not libre_ant) and salida_libre and load_tarea;
    load_tarea_i <= salida_puertaAND3;

    libre_sig <= libre_ant or salida_libre;

end iterativaBasicaArch;
```

EntradaTablaTareas.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity entradaTablaTareas is
    generic( numBitsContador : integer := 3;
                numBitsEtiqueta : integer := 3);
    port (
        clk, rst, loadTarea, dependenciaResuelta : in std_logic;
        prepararaParaEjecutar, acierto : out std_logic;
        etiquetaIn : in std_logic_vector(numBitsEtiqueta-1 downto 0);
        --LA ENTRADA ES: Etiqueta--numSucesores--numPredecesores--subtareas
        entradaDatos : in std_logic_vector
                    ( numBitsContador+numBitsEtiqueta*(2**numBitsContador)
                                                +numBitsEtiqueta-1 downto 0);
        --LA SALIDA ES: numSucesores--subtareas
        salidaDatos : out std_logic_vector(numBitsEtiqueta
                      +numBitsEtiqueta*(2**numBitsContador-1)-1 downto 0) );
end entradaTablaTareas;

architecture entradaTablaTareasArch of entradaTablaTareas is

component registro
   generic( n : integer := numBitsEtiqueta );
   port( clk, rst, ld : in std_logic;
      din : in std_logic_vector( n-1 downto 0 );
      dout : out std_logic_vector( n-1 downto 0 ) );
end component;

component contador
   generic( n : integer := numBitsContador );
   port( clk, rst, ld, dec : in std_logic;
      din : in std_logic_vector( n-1 downto 0 );
      dout : out std_logic_vector( n-1 downto 0 ) );
end component;


  signal salidaEtiquetaEntrada:
                          std_logic_vector (numBitsEtiqueta-1 downto 0);
  signal salidaContadorPredecesores, cero:
                          std_logic_vector (numBitsContador-1 downto 0);
  signal decrementar: std_logic;
```

```
    signal salidaComparador, hayCeroPredecesores: std_logic;

  begin

    cero <= (others =>'0');

    etiquetaEntrada: registro generic map (numBitsEtiqueta)
                     port map (clk=>clk,
                               rst=>rst,
                               ld=>loadTarea,
                               din=>
                                 entradaDatos(numBitsContador+numBitsEtiqueta*
                                 (2**numBitsContador)+numBitsEtiqueta-1
                                 downto
                                 numBitsContador+numBitsEtiqueta*(2**
                                 numBitsContador)),
                               dout=>salidaEtiquetaEntrada);

    registroNumSucesores: registro generic map (numBitsContador)
                     port map (clk=>clk,
                               rst=>rst,
                               d=>loadTarea,din=>entradaDatos(numBitsContador+
                                   numBitsEtiqueta*(2**numBitsContador)-1
                                   downto
                                   numBitsContador+numBitsEtiqueta*(2**
                                   numBitsContador-1)),
                               dout=>salidaDatos(
                                  numBitsEtiqueta+numBitsEtiqueta*
                                  (2**numBitsContador-1)-1
                                  downto
                                   numBitsEtiqueta*(2**numBitsContador-1)) );

    contadorPredecesores: contador generic map (numBitsContador)
                          port map (clk=>clk,
                                    rst=>rst,
                                    ld=>loadTarea,
                                    dec=>decrementar,
                                    din=>entradaDatos(numBitsContador+
                                        numBitsEtiqueta*(2**numBitsContador
                                        -1)-1
                                        downto
                                        numBitsEtiqueta*(2**numBitsContador-1)),
                                    dout=>salidaContadorPredecesores);


    subtareas : for I in 0 to 2**numBitsContador-2 generate
    subtarea : registro generic map (numBitsEtiqueta)
                  port map (clk=>clk,
                            rst=>rst,
                            ld=>loadTarea,
                            din=>entradaDatos(I*numBitsEtiqueta+
                                  numBitsEtiqueta-1 downto I*numBitsEtiqueta),
                            dout=>salidaDatos(I*numBitsEtiqueta+
                                  numBitsEtiqueta-1 downto I*numBitsEtiqueta));
    end generate;


    salidaComparador <= '1' when (salidaEtiquetaEntrada = etiquetaIn) else '0';
    hayCeroPredecesores <= '1' when (salidaContadorPredecesores = cero)
                                                              else '0';

    prepararaparaejecutar <=  hayCeroPredecesores and salidaComparador;
    acierto <= salidaComparador;
    decrementar <= salidaComparador and dependenciaResuelta;

  end entradaTablaTareasArch;
```

TablaTareas.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity tablatareas is
    generic( numbitscontador : integer := 3;
             numbitsetiqueta : integer := 3;
             numentradaslog : integer := 3 );
    port (
        clk, rst, loadtarea, actualizardependencias, borrar : in std_logic;
        prepararaparaejecutar, acierto, tablallena : out std_logic;
        etiquetain : in std_logic_vector(numbitsetiqueta-1 downto 0);
        entradadatos : in
                        std_logic_vector(numbitscontador+numbitsetiqueta*(2*
                        *numbitscontador)+numbitsetiqueta-1 downto 0);
        salidadatos : out
                        std_logic_vector(numbitsetiqueta+numbitsetiqueta*(2*
                        *numbitscontador-1)-1 downto 0);
        numentradasocupadas : out std_logic_vector(numentradaslog downto 0));
end tablatareas;

architecture tablatareasarch of tablatareas is


component entradatablatareas
    generic( numbitscontador : integer := 3;
             numbitsetiqueta : integer := 3 );
    port (
        clk, rst, loadtarea, dependenciaresuelta : in std_logic;
        prepararaparaejecutar, acierto : out std_logic;
        etiquetain : in std_logic_vector(numbitsetiqueta-1 downto 0);
        entradadatos : in
                std_logic_vector(numbitscontador+numbitsetiqueta*(2**numbits
                contador)+numbitsetiqueta-1 downto 0);
        salidadatos : out
                std_logic_vector(numbitsetiqueta+numbitsetiqueta*(2**numbits
                contador-1)-1 downto 0) );
end component;

component priorityencoder
    generic( n : integer := numentradaslog );
    port(
       x : in std_logic_vector( 2**n-1 downto 0 );
       y : out std_logic_vector ( n-1 downto 0 );
       gs : out std_logic );
end component;

component iterativabasica
   port( clk,rst,libre_ant, borrar, acierto_i, load_tarea : in std_logic;
       load_tarea_i,libre_sig : out std_logic);
end component;

component contadorincdec
   generic( n : integer := 8 );
   port( clk, rst, inc, dec : in std_logic;
      dout : out std_logic_vector( n-1 downto 0 ) );
end component;

  signal preparadas, aciertos: std_logic_vector
                                        ((2**numentradaslog)-1 downto 0);
  signal libre: std_logic_vector ((2**numentradaslog) downto 0);
  signal loadtareain: std_logic_vector ((2**numentradaslog)-1 downto 0);
  signal salidacodificador: std_logic_vector (numentradaslog-1 downto 0);
  signal salidasdatos: std_logic_vector
                  ((numbitsetiqueta+numbitsetiqueta*(2**numbitscontador-1))
```

```
                    *(2**numentradaslog)-1 downto 0);
   signal salidadatosintermedia:std_logic_vector
                    (numbitsetiqueta+numbitsetiqueta*(2**numbitscontador-1)-1
                     downto 0);
   signal cero: std_logic_vector (numbitscontador-1 downto 0);
   signal algunavaleuno: std_logic;

begin

    cero <= (others=>'0');

    tabla : for i in 0 to (2**numentradaslog)-1 generate
       subtarea : entradatablatareas
            generic map (numbitscontador, numbitsetiqueta)
             port map(clk=>clk,
                      rst=>rst,
                      loadtarea=>loadtareain(i),
                      dependenciaresuelta=>actualizardependencias,
                      prepararaparaejecutar=>preparadas(i),
                      acierto=>aciertos(i),
                      etiquetain=>etiquetain,
                      entradadatos=>entradadatos,
                      salidadatos=>
                          salidasdatos((numbitsetiqueta+numbitsetiqueta*
                          (2**numbitscontador-1))
                          *i+(numbitsetiqueta+numbitsetiqueta*
                          (2**numbitscontador-1))-1
                          downto
                          (numbitsetiqueta+numbitsetiqueta*
                          (2**numbitscontador-1))*i ) );
    end generate;

    libre(0) <= '0';

    rediterativa : for i in 0 to (2**numentradaslog)-1 generate
       celdabasica: iterativabasica
          port map ( clk, rst, libre(i), borrar, aciertos(i), loadtarea,
                     loadtareain(i), libre(i+1) );
    end generate;

    prepararaparaejecutar <= '0' when (preparadas = cero) else '1';

    codificador: priorityencoder generic map (numentradaslog)
          port map (aciertos, salidacodificador, algunavaleuno);

    salidadatosintermedia <= salidasdatos(
                 (numbitsetiqueta*(2**numbitscontador-1)+numbitsetiqueta)
                 *conv_integer(salidacodificador)+numbitsetiqueta*(2**
                 numbitscontador-1)+numbitsetiqueta-1
                 downto
                 (numbitsetiqueta*(2**numbitscontador-1)+numbitsetiqueta)*
                 conv_integer(salidacodificador) );

    salidadatos <= salidadatosintermedia;
    tablallena <= not libre(2**numentradaslog);
    acierto <= '0' when (aciertos = cero) else '1';

    contadorentradasocupadas : contadorincdec generic map (numentradaslog+1)
       port map (clk=>clk,
                 rst=>rst,
                 inc=>loadtarea, --se debe controlar fuera que no este llena
                 dec=>borrar, --se controla fuera que el borrado sea correcto
                 dout=>numentradasocupadas );

end tablatareasarch;
```

FifoControlTablaTareas.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity fifoControlTablaTareas is
    generic( numBitsContador : integer := 3;
                numBitsEtiqueta : integer := 3;
                numEntradasLogTabla : integer := 3;
                numEntradasLogFifo : integer := 3 );
    port (
        clk, rst, borrar : in std_logic;
        prepararaParaEjecutar, acierto,
                 preparadaTablaSiguienteOperacion : out std_logic;
        etiquetaIn : in std_logic_vector(numBitsEtiqueta-1 downto 0);
        entradaDatosFifo : in
                 std_logic_vector(numBitsContador+numBitsEtiqueta*(2**
                 numBitsContador-1)+numBitsEtiqueta+numBitsEtiqueta-1
                 downto 0);
        escribirFifo: in std_logic;
        fifoSubtareasLlena: out std_logic );
end fifoControlTablaTareas;

architecture fifoControlTablaTareasArch of fifoControlTablaTareas is

component tablaTareas
    generic( numBitsContador : integer := 3;
                numBitsEtiqueta : integer := 3;
                numEntradasLog : integer := 3 );
    port (
        clk, rst, loadTarea, actualizarDependencias, borrar : in std_logic;
        prepararaParaEjecutar, acierto, tablaLlena : out std_logic;
        etiquetaIn : in std_logic_vector(numBitsEtiqueta-1 downto 0);
        entradaDatos : in
           std_logic_vector(numBitsContador+numBitsEtiqueta*(2**
                              numBitsContador)+numBitsEtiqueta-1 downto 0);
         salidaDatos : out
           std_logic_vector(numBitsEtiqueta+numBitsEtiqueta*(2**
                              numBitsContador-1)-1 downto 0);
        numEntradasOcupadas : out std_logic_vector(numEntradasLog downto 0));
end component;

component registro
   generic( n : integer := numBitsEtiqueta );
   port( clk, rst, ld : in std_logic;
      din : in std_logic_vector( n-1 downto 0 );
      dout : out std_logic_vector( n-1 downto 0 ) );
end component;

component contador
   generic( n : integer := numBitsContador );
   port( clk, rst, ld, dec : in std_logic;
      din : in std_logic_vector( n-1 downto 0 );
      dout : out std_logic_vector( n-1 downto 0 ) );
end component;

component fifo30Bits
     generic (numEntradasLog : integer := 3 );
     port (entradaDatos: in std_logic_vector(29 downto 0);
                  clk, rst, leer, escribir, enable : in std_logic;

     salidaDatos: out std_logic_vector(29 downto 0);
     fifoLlena, fifoVacia: out std_logic);
end component;

  type states is (reposo, esperando, cargaSubtareaPrimera,
```

```
      cargaSubtareaContinua, cargaSubtareaFinal, lecturaPrimeraPalabraCarga,
      lecturaPrimeraPalabraLectura);
   signal currentState, nextState: states;

   type statesConsultaActualizarDependencias is
                  (reposoConsultaActualizarDependencias, carga, actualizar);
   signal currentStateConsultaActualizarDependencias,
         nextStateConsultaActualizarDependencias:
         statesConsultaActualizarDependencias;

   signal etiqueta : std_logic_vector(numBitsEtiqueta-1 downto 0);
   signal salidaContador, ceroNumBitsContador, unoNumBitsContador:
                        std_logic_vector (numBitsContador-1 downto 0);
   signal ceroNumEntradasLogFifo, unoNumEntradasLogFifo :
                        std_logic_vector (numEntradasLogFifo-1 downto 0);
   signal salidaRegistroAumentada :
                        std_logic_vector (numBitsEtiqueta*(2**
                                          numBitsContador)-1 downto 0);
   signal decrementarContador, dependenciaResuelta, loadContadorRegistro,
                                          borrarControlador: std_logic;
   signal salidaDatosIntermedia: std_logic_vector(numBitsEtiqueta+
                        numBitsEtiqueta*(2**numBitsContador-1)-1 downto 0);
   signal numEntradasOcupadas : std_logic_vector(numEntradasLogTabla
                                                          downto 0);
   signal aciertoIntermedia : std_logic;
   signal fifoLlena, fifoVacia, enableFifo, leerFifoMaquinaEstados,
                               escribirFifoControl, loadTarea : std_logic;
   signal salidaDatosFifo : std_logic_vector(numBitsContador+numBitsEtiqueta
                                *(2**numEntradasLogTabla-1)+
                                numBitsEtiqueta+numBitsEtiqueta-1 downto 0);

   signal tablaLlena: std_logic;

   signal salidaNumeroNodos: std_logic_vector(numEntradasLogFifo-1 downto 0);
   signal loadContadorNumeroNodos, decrementarNumeroNodos, uno : std_logic;
   signal salidaRegistroEtiqueta: std_logic_vector(numBitsEtiqueta-1
                                                          downto 0);

begin

    uno <= '1';

    ceroNumBitsContador <= (others=>'0');
    unoNumBitsContador <= ceroNumBitsContador+1;

    ceroNumEntradasLogFifo <= (others=>'0');
    unoNumEntradasLogFifo <= ceroNumEntradasLogFifo+1;

    fifoSubtareasLlena <= fifoLlena;
    escribirFifoControl <= escribirFifo and not fifoLlena;
    enableFifo <= '1';

    fifoTabla : fifo30Bits generic map(numEntradasLogFifo)
                     port map (entradaDatos=>entradaDatosFifo,
                     clk=>clk,
                     rst=>rst,
                     leer=>leerFifoMaquinaEstados,
                     escribir=>escribirFifoControl,
                     enable=>enableFifo,
                     salidaDatos=>salidaDatosFifo,
                     fifoLlena=>fifoLlena,
                     fifoVacia=>fifoVacia );


    contadorNumeroNodos: contador
            generic map (numEntradasLogFifo)
             port map (clk=>clk,
             rst=>rst,
```

```
            ld=>loadContadorNumeroNodos,
            dec=>decrementarNumeroNodos,
            din=>salidaDatosFifo(numBitsContador+
               numBitsEtiqueta*(2** numEntradasLogTabla-)+numBitsEtiqueta+
               numBitsEtiqueta-1
               downto
               numBitsContador+numBitsEtiqueta* (2**numEntradasLogTabla-1)+
               numBitsEtiqueta+numBitsEtiqueta- numEntradasLogFifo),
            dout=>salidaNumeroNodos);

      tabla : tablaTareas
         generic map (numBitsContador, numBitsEtiqueta, numEntradasLogTabla)
         port map(clk=>clk,
                  rst=>rst,
                  loadTarea=>loadTarea,
                  actualizarDependencias=>dependenciaResuelta,
                  borrar=> borrarControlador,
                  prepararaParaEjecutar=> prepararaParaEjecutar,
                  acierto=>aciertoIntermedia,
                  tablaLlena=> tablaLlena,
                  etiquetaIn=>etiqueta,
                  entradaDatos=>salidaDatosFifo
                  salidaDatos=>salidaDatosIntermedia,
                  numEntradasOcupadas=>numEntradasOcupadas );
                  acierto <= aciertoIntermedia;

      --Control de la tabla
      stateGen:
           process (escribirFifo, currentState, salidaNumeroNodos,
                    numEntradasOcupadas, fifoVacia, tablaLlena)
      begin
        nextState <= currentState;
        case currentState is

       when reposo =>
           if (salidaNumeroNodos = ceroNumEntradasLogFifo and fifoVacia='0')
                           then nextState <= lecturaPrimeraPalabraLectura;
           end if;

       when lecturaPrimeraPalabraLectura =>
           if (escribirFifo='0') then
              nextState <= lecturaPrimeraPalabraCarga;
           end if;

       when lecturaPrimeraPalabraCarga =>
           if (2**numEntradasLogTabla-numEntradasOcupadas >=
               salidaNumeroNodos  and salidaNumeroNodos /=
               ceroNumEntradasLogFifo and fifoVacia='0' and
               tablaLlena='0') then
                     nextState <= cargaSubtareaPrimera;
           else
               nextState <= esperando;
           end if;

             when esperando =>
           if (2**numEntradasLogTabla-numEntradasOcupadas >=
               salidaNumeroNodos and salidaNumeroNodos /=
                ceroNumEntradasLogFifo and fifoVacia='0' and tablaLlena='0')
                then nextState <= cargaSubtareaPrimera;
           end if;

              when cargaSubtareaPrimera =>
            if (escribirFifo='0') then
                if (salidaNumeroNodos = unoNumEntradasLogFifo) then
                     nextState <= cargaSubtareaFinal;
                else
                     nextState <= cargaSubtareaContinua;
                end if;
```

```
        end if;

          when cargaSubtareaContinua =>
            if (escribirFifo='0') then
               if (salidaNumeroNodos = unoNumEntradasLogFifo) then
                   nextState <= cargaSubtareaFinal;
               end if;
            end if;

           when cargaSubtareaFinal => nextState <= reposo;

 end case;
end process stateGen;

state:
process (rst, clk)
begin
  if (rst = '1') then
    currentState <= reposo;
  elsif (clk'event and clk='1') then
    currentState <= nextState;
  end if;
end process state;

mooreGen:
process (escribirFifo, currentState)
begin
  case currentState is

 when reposo =>
    loadTarea <='0';
    decrementarNumeroNodos <= '0';
    leerFifoMaquinaEstados <= '0';
          loadContadorNumeroNodos <= '0';

      when esperando =>
    loadTarea <='0';
    decrementarNumeroNodos <= '0';
    leerFifoMaquinaEstados <= '0';
          loadContadorNumeroNodos <= '0';

 when cargaSubtareaPrimera =>
   loadTarea <='0';
   decrementarNumeroNodos <= not escribirFifo;
   leerFifoMaquinaEstados <= not escribirFifo;
         loadContadorNumeroNodos <= '0';

 when cargaSubtareaContinua =>
   loadTarea <= not escribirFifo;
   decrementarNumeroNodos <= not escribirFifo;
   leerFifoMaquinaEstados <= not escribirFifo;
         loadContadorNumeroNodos <= '0';

 when cargaSubtareaFinal =>
   loadTarea <='1';
   decrementarNumeroNodos <= '0';
   leerFifoMaquinaEstados <= '0';
         loadContadorNumeroNodos <= '0';

 when lecturaPrimeraPalabraLectura =>
   loadTarea <='0';
   decrementarNumeroNodos <= '0';
   leerFifoMaquinaEstados <= not escribirFifo;
         loadContadorNumeroNodos <= '0';

 when lecturaPrimeraPalabraCarga =>
   loadTarea <='0';
   decrementarNumeroNodos <= '0';
```

```
        leerFifoMaquinaEstados <= '0';
              loadContadorNumeroNodos <= '1';

     end case;
    end process mooreGen;


    -- Esto es para actualizar las dependencias
    stateGenConsultaActualizarDependencias:
    process (currentStateConsultaActualizarDependencias, borrar,
                  salidaContador, salidaDatosIntermedia, aciertoIntermedia)
    begin
      nextStateConsultaActualizarDependencias <=
                               currentStateConsultaActualizarDependencias;
      case currentStateConsultaActualizarDependencias is

        when reposoConsultaActualizarDependencias =>
          if (borrar ='1'  and aciertoIntermedia ='1') then
               nextStateConsultaActualizarDependencias <= carga;
          end if;

        when carga =>
          if (salidaDatosIntermedia(numBitsEtiqueta*(2**numBitsContador-
             1)+numBitsEtiqueta-1 downto
             numBitsEtiqueta*(2**numBitsContador-1)+numBitsEtiqueta-
             numBitsContador) = ceroNumBitsContador)
             then
               nextStateConsultaActualizarDependencias <=
               reposoConsultaActualizarDependencias;
             else
               nextStateConsultaActualizarDependencias <= actualizar;
             end if;

        when actualizar =>
          if (salidaContador = unoNumBitsContador) then
             nextStateConsultaActualizarDependencias <=
             reposoConsultaActualizarDependencias;
          end if;

      end case;
    end process stateGenConsultaActualizarDependencias;

     stateConsultaActualizarDependencias: process (rst, clk)
     begin
       if (rst = '1') then
         currentStateConsultaActualizarDependencias <=
                               reposoConsultaActualizarDependencias;
       elsif (clk'event and clk='1') then
         currentStateConsultaActualizarDependencias <=
                               nextStateConsultaActualizarDependencias;
       end if;
     end process stateConsultaActualizarDependencias;

     mooreMealyGenConsultaActualizarDependencias:
     process (currentStateConsultaActualizarDependencias,
             etiquetaIn, salidaRegistroAumentada, salidaContador,
             salidaDatosFifo, borrar, aciertoIntermedia,
             salidaRegistroEtiqueta)
     begin
       case currentStateConsultaActualizarDependencias is

       when reposoConsultaActualizarDependencias =>
             preparadaTablaSiguienteOperacion <= '1';
             borrarControlador <='0';
             dependenciaResuelta <= '0';
             decrementarContador <= '0';
             etiqueta <= salidaRegistroEtiqueta;
```

```
        when carga =>
             preparadaTablaSiguienteOperacion <= '0';
             borrarControlador <='1';
             dependenciaResuelta <= '0';
             loadContadorRegistro <= '1';
             decrementarContador <= '0';
             etiqueta <= salidaRegistroEtiqueta;

        when actualizar =>
             preparadaTablaSiguienteOperacion <= '0';
             borrarControlador <='0';
             dependenciaResuelta <= '1';
             loadContadorRegistro <= '0';
             decrementarContador <= '1';
             etiqueta <= salidaRegistroAumentada( (numBitsEtiqueta*
                    (CONV_INTEGER(salidaContador))+numBitsEtiqueta-1) downto
                    (numBitsEtiqueta*(CONV_INTEGER(salidaContador))));
      end case;
     end process mooreMealyGenConsultaActualizarDependencias;


    registroEtiqueta :  registro generic map (numBitsEtiqueta)
                 port map (clk=>clk,
                           rst=>rst,
                           ld=>uno,
                           din=>etiquetaIn,
                           dout=>salidaRegistroEtiqueta);

      registroSucesores :  registro
                generic map (numBitsEtiqueta*(2**numBitsContador-1))
                port map (clk=>clk,
                          rst=>rst,
                          ld=>loadContadorRegistro,
                          din=>salidaDatosIntermedia(numBitsEtiqueta*
                                (2**numBitsContador-1)-1 downto 0),
                          dout=>salidaRegistroAumentada(numBitsEtiqueta*
                                (2**numBitsContador)-1 downto
                                numBitsEtiqueta));

      salidaRegistroAumentada (numBitsEtiqueta-1 downto 0) <= (others=>'0');

      contadorSucesores: contador
                 generic map (numBitsContador)
                port map (clk=>clk,
                          rst=>rst,
                          ld=>loadContadorRegistro,
                          dec=>decrementarContador,
                          din=>salidaDatosIntermedia(
                               numBitsEtiqueta*(2**numBitsContador-1)+
                               numBitsEtiqueta-1
                               downto
                               numBitsEtiqueta*(2**numBitsContador-1)+
                               numBitsEtiqueta-numBitsContador),
                          dout=>salidaContador);


  end fifoControlTablaTareasArch;
```

TablaURControlFifoEventos.vhd

```
  library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_arith.all;
  use IEEE.std_logic_unsigned.all;

  entity tablaUFControlFifoEventos is
```

```
    generic( numBitsContadorTablaTareas : integer := 3;
             numBitsEtiqueta : integer := 3;
             numEntradasLogTablaTareas : integer := 3;
             numEntradasLogFifoTablaTareas : integer := 3;
             numUFLog : integer := 2;
             numEntradasLogFifoEventos : integer := 3;
             numEntradasLogFifoUFs : integer := 3;
             numEntradasLogFifoReconfiguraciones : integer := 3 );

    port ( clk, rst : in std_logic;
           entradaDatosFifoTablaTareas : in
             std_logic_vector(numBitsContadorTablaTareas+numBitsEtiqueta*
                      (2**numEntradasLogTablaTareas-1)+numBitsEtiqueta+
                       numBitsEtiqueta-1 downto 0);
           escribirFifoTablaTareas: in std_logic;
           fifoSubtareasLlena: out std_logic;
           entradaDatosFifoUFs: in
              std_logic_vector((2**numUFLog)*(2*numBitsEtiqueta+2*5)-1
                                                              downto 0);
           escribirFifoUFs: in std_logic_vector (2**numUFLog-1 downto 0);
           fifosUFLlenas: out std_logic_vector (2**numUFLog-1 downto 0);
           UFenables: in std_logic_vector (2**numUFLog-1 downto 0);
           entradaDatosFifoReconfiguraciones: in
                  std_logic_vector(numBitsEtiqueta-1 downto 0);
           escribirFifoReconfiguraciones: in std_logic;
           fifoReconfiguracionesFifoLlena: out std_logic;
           fifoReconfiguracionesenable: in std_logic;
           circuitoReconfiguracionLibre : in std_logic;
           eventoNuevaTarea: in std_logic;
           concedidaEscrituraNuevaTarea: out std_logic );
end tablaUFControlFifoEventos;

architecture tablaUFControlFifoEventosArch of tablaUFControlFifoEventos is

component fifoControlTablaTareas
    generic( numBitsContador : integer := 3;
                  numBitsEtiqueta : integer := 3;
                  numEntradasLogTabla : integer := 3;
             numEntradasLogFifo : integer := 3 );
    port (clk, rst, borrar : in std_logic;
          prepararaParaEjecutar, acierto,
                  preparadaTablaSiguienteOperacion : out std_logic;
          etiquetaIn : in std_logic_vector(numBitsEtiqueta-1 downto 0);
          entradaDatosFifo : in
                    std_logic_vector(numBitsContador+numBitsEtiqueta*
                                    (2**numEntradasLogTabla-1)+
                                     numBitsEtiqueta+numBitsEtiqueta-1
                                     downto 0);
          escribirFifo: in std_logic;
          fifoSubtareasLlena: out std_logic );
end component;

component UR
    generic (numBitsEtiqueta:natural:=3;
                  longitud:natural:=30;
                  numEntradasLog:natural:=3;
                  long_eje_carga:natural:=5);
    port ( clk: in std_logic;
           rst: in std_logic;
           enable: in std_logic;
           entrada_fifo:in std_logic_vector(longitud-1 downto 0);
           escribir: in std_logic;
           comenzar_carga : in std_logic;
           comenzar_ejecucion : in std_logic;
           pedir_carga : out std_logic;
           pedir_ejecucion : out std_logic;
           sub_actual : out std_logic_vector(longitud-1 downto 0);
           tabla_llena: out std_logic;
```

```
            estado: out std_logic_vector(2 downto 0);
            vacia: out std_logic;
            escribirEnFifoEventos: out std_logic;
            evento: out std_logic_vector(numBitsEtiqueta+2-1 downto 0);
            concedidaEscritura: in std_logic);
end component;

component fifo3Bits
     generic (numEntradasLog : integer := 3 );
     port (entradaDatos: in std_logic_vector(2 downto 0);
                 clk, rst, leer, escribir, enable : in std_logic;
                 salidaDatos: out std_logic_vector(2 downto 0);
                 fifoLlena, fifoVacia: out std_logic);
end component;

component fifo5Bits
     generic (numEntradasLog : integer := 3 );
     port (entradaDatos: in std_logic_vector(4 downto 0);
                 clk, rst, leer, escribir, enable : in std_logic;
                 salidaDatos: out std_logic_vector(4 downto 0);
                 fifoLlena, fifoVacia: out std_logic);
end component;

component priorityEncoder
    generic( n : integer := 3 );
    port(
       x : in std_logic_vector( 2**n-1 downto 0 );
       y : out std_logic_vector ( n-1 downto 0 );
       gs : out std_logic );
end component;

component decoder
    generic( n : integer := 3 );
    port(
       x : in std_logic_vector( n-1 downto 0 );
       en : in std_logic;
       y : out std_logic_vector ( 2**n-1 downto 0 ) );
end component;

component contadorInc
   generic( n : integer := 8 );
   port( clk, rst, inc : in std_logic;
      dout : out std_logic_vector( n-1 downto 0 ) );
end component;

--Señales para la tabla de tareas
  signal tablaTareasBorrar, tablaTareasPrepararaParaEjecutar,
         tablaTareasAcierto,
         tablaTareasPreparadaTablaSiguienteOperacion : std_logic;
  signal tablaTareasEtiquetaIn : std_logic_vector(numBitsEtiqueta-1
                                                          downto 0);

  --Señales para las UR
  signal UFcomenzar_cargas, UFcomenzar_ejecuciones, UFpedir_cargas,
         UFpedir_ejecuciones, UFvacia, UFescribirEnFifoEventos,
         UFconcedidaEscritura : std_logic_vector (2**numUFLog-1 downto 0);
  signal UFeventos: std_logic_vector((numBitsEtiqueta+2)*(2**numUFLog)-1
                                                          downto 0);
  signal UFestados: std_logic_vector(3*(2**numUFLog)-1 downto 0);
  signal UFsub_actuales :
          std_logic_vector((2**numUFLog)*(2*numBitsEtiqueta+2*5)-1 downto 0);

  --Señales para la fifo de eventos
  signal fifoEventosEntradaDatos, fifoEventosSalidaDatos:
         std_logic_vector(2+numBitsEtiqueta-1 downto 0);
  signal fifoEventosenable, fifoEventosLeer, fifoEventosEscribir,
         fifoEventosFifoLlena, fifoEventosFifoVacia : std_logic;
```

```
    --Señales para la fifo de reconfiguraciones
    signal fifoReconfiguracionesSalidaDatos:
                                    std_logic_vector(numBitsEtiqueta-1 downto
                                    0);
    signal fifoReconfiguracionesLeer, fifoReconfiguracionesFifoVacia :
                                    std_logic;

    --Señales para el codificador de la subtarea a cargar
    signal iguales, igualesEvento: std_logic_vector ((2**numUFLog)-1 downto 0);
    signal codificadorSubtareaACargarSalida :
                                    std_logic_vector (numUFLog-1 downto 0);
    signal codificadorSubtareaACargarAlgunaValeUno,
                                    subtareaACargarComenzarCarga : std_logic;

    --Señales para el codificador de comprobar dependencias
    signal codificadorComprobarDependenciasSalida :
                                    std_logic_vector (numUFLog-1 downto 0);
    signal codificadorComprobarDependenciasAlgunaValeUno,
                            comprobarDependenciasComenzarEjecucion : std_logic;

    --Señales para el controlador
    type states is (reposoOLectura, distinguirEvento,
       lecturaFifoReconfiguracionesEventoNuevaTarea,
       distinguirEstadoUnidadEventoNuevaTarea,
       envioComenzarCargaEventoNuevaTarea,
       comprobarDependenciasEventoFinReconfiguracion,
       subtareaPreparadaEventoFinReconfiguracion,
       envioComenzarEjecucionEventoFinReconfiguracion, actualizarDependencias,
       lecturaFifoReconfiguracionesEventoFinEjecucion,
       distinguirEstadoUnidadEventoFinEjecucion,
       envioComenzarCargaEventoFinEjecucion, bucleFor,
       comprobarDependenciasEventoFinEjecucion,
       subtareaPreparadaEventoFinEjecucion,
       envioComenzarEjecucionEventoFinEjecucion);
    signal currentState, nextState: states;

    --Señales para el contador del bucle
    signal contadorBucleIncremento, bucle, rstContadorBucle: std_logic;
    signal contadorBucleSalida: std_logic_vector (numUFLog-1+1 downto 0);

    signal finEjecucion, finReconfiguracion, nuevaTarea:
                                            std_logic_vector (1 downto 0);

    signal contadorBucleSalidaMayorNumUF :
                                    std_logic_vector (numUFLog-1+1 downto 0);
    signal cerosNumBitsEtiqueta :
                            std_logic_vector (numBitsEtiqueta-1 downto 0);

    --Señales para el biestable RS deboLeerFifoReconfiguracion
    signal deboLeerReset, deboLeerSet, deboLeerSalida : std_logic;

    --Para seleccionar a que unidad enviar el comenzar ejecución
    signal decodificadorComprobarDependenciasEntrada:
                            std_logic_vector (numUFLog-1 downto 0);

    signal contadorBucleSalidaMenosUno: std_logic_vector (numUFLog-1 downto 0);

  begin

     tablaTareas : fifoControlTablaTareas
         generic map( numBitsContadorTablaTareas, numBitsEtiqueta,
                      numEntradasLogTablaTareas, numEntradasLogFifoTablaTareas)
         port map (clk=>clk,
                   rst=>rst,
                   borrar=>tablaTareasBorrar,
                   prepararaParaEjecutar=>tablaTareasPrepararaParaEjecutar,
                   acierto=>tablaTareasAcierto,
                   preparadaTablaSiguienteOperacion=>
```

```
                                tablaTareasPreparadaTablaSiguienteOperacion,
                etiquetaIn=>tablaTareasEtiquetaIn,
                entradaDatosFifo=>entradaDatosFifoTablaTareas,
                escribirFifo=>escribirFifoTablaTareas,
                fifoSubtareasLlena=>fifoSubtareasLlena);

    UFs : for I in 0 to (2**numUFLog)-1 generate
          unidadFuncional: UF
           generic map (numBitsEtiqueta, 2*numBitsEtiqueta+2*5,
                                                numEntradasLogFifoUFs, 5)
           port map ( clk=>clk,
                      rst=>rst,
                      enable=>UFenables(I),
                      entrada_fifo=>entradaDatosFifoUFs(I*(2*
                      numBitsEtiqueta+2*5)+(2*numBitsEtiqueta+2*5)-
                      1 downto I*(2*numBitsEtiqueta+2*5)),
                      escribir=>escribirFifoUFs(I),
                      comenzar_carga=>UFcomenzar_cargas(I),
                      comenzar_ejecucion=>UFcomenzar_ejecuciones(I),
                      pedir_carga=>UFpedir_cargas(I),
                      pedir_ejecucion=>UFpedir_ejecuciones(I),
                      sub_actual=> UFsub_actuales(I*(2*numBitsEtiqueta+2*5)
                                                  +2*numBitsEtiqueta+2*5-1
                                                  downto
                                                  I*(2*numBitsEtiqueta+2*5)),
                      tabla_llena=>fifosUFLlenas(I),
                      estado=>UFestados(I*3+3-1 downto I*3),
                      vacia=>UFvacia(I),
                      escribirEnFifoEventos=>UFescribirEnFifoEventos(I),
                      evento=>UFeventos(I*(numBitsEtiqueta+2)+(numBitsEtiqueta
                                        +2)-1 downto I*(numBitsEtiqueta+2)),
                      concedidaEscritura=>UFconcedidaEscritura(I) );

  iguales(I)<= '1' when
                   UFsub_actuales(I*(2*numBitsEtiqueta+2*5)+numBitsEtiqueta+2*
                   5-1 downto
                   I*(2*numBitsEtiqueta+2*5)+2*5) =
                   fifoReconfiguracionesSalidaDatos else '0';

  igualesEvento(I) <= '1' when
                   UFsub_actuales(I*(2*numBitsEtiqueta+2*5)+2*numBitsEtiqueta+
                   2*5-1 downto I*(2*numBitsEtiqueta+2*5)+numBitsEtiqueta+2*5)
                   = fifoEventosSalidaDatos(2+numBitsEtiqueta-1 downto 2)
                   else '0';

  end generate;



  fifoEventosenable <= '1';

  fifoEventos: fifo5Bits generic map (numEntradasLogFifoEventos)
               port map (entradaDatos=>fifoEventosEntradaDatos,
                         clk=>clk,
                         rst=>rst,
                         leer=>fifoEventosLeer,
                         escribir=>fifoEventosEscribir,
                         enable=>fifoEventosenable,
                         salidaDatos=>fifoEventosSalidaDatos,
                         fifoLlena=>fifoEventosFifoLlena,
                         fifoVacia=>fifoEventosFifoVacia);

   fifoReconfiguraciones: fifo3Bits generic map
                                 (numEntradasLogFifoReconfiguraciones)
               port map (entradaDatos=>entradaDatosFifoReconfiguraciones,
                         clk=>clk,
                         rst=>rst,
                         leer=>fifoReconfiguracionesLeer,
```

```
                        escribir=>escribirFifoReconfiguraciones,
                        enable=>fifoReconfiguracionesenable,
                        salidaDatos=>fifoReconfiguracionesSalidaDatos,
                        fifoLlena=>fifoReconfiguracionesFifoLlena,
                        fifoVacia=>fifoReconfiguracionesFifoVacia);

  tablaTareasEtiquetaIn <=
       fifoEventosSalidaDatos(numBitsEtiqueta+2-1 downto 2) when bucle='0'
       else
       UFsub_actuales(conv_integer(contadorBucleSalida(numUFLog-1 downto 0))*
       (2*numBitsEtiqueta+2*5)+(2*numBitsEtiqueta+2*5)-1 downto
       conv_integer (contadorBucleSalida(numUFLog-1 downto 0))
       *(2*numBitsEtiqueta+2*5)+numBitsEtiqueta+2*5 );

  --Controlador y elementos para el mismo

   contadorBucle:  contadorInc generic map (numUFLog+1)
                         port map(clk=>clk,
                                  rst=>rstContadorBucle,
                                  inc=>contadorBucleIncremento,
                                  dout=>contadorBucleSalida );

   decodificadorSubtareaACargar: decoder generic map (numUFLog)
                         port map(x=>codificadorSubtareaACargarSalida,
                                  en=>subtareaACargarComenzarCarga,
                                  y=>UFcomenzar_cargas);

    codificadorSubtareaACargar: priorityEncoder generic map (numUFLog)
                         port map(x=>iguales,
                                  y=>codificadorSubtareaACargarSalida,
                                  gs=>codificadorSubtareaACargarAlgunaValeUno);

   decodificadorComprobarDependencias: decoder generic map (numUFLog)
                         port map(x=>decodificadorComprobarDependenciasEntrada,
                                  en=>comprobarDependenciasComenzarEjecucion,
                                  y=>UFcomenzar_ejecuciones);

   codificadorComprobarDependencias: priorityEncoder generic map (numUFLog)
                         port map(x=>igualesEvento,
                                  y=>codificadorComprobarDependenciasSalida,
                            gs=>codificadorComprobarDependenciasAlgunaValeUno);

   finEjecucion<="10";
   finReconfiguracion<="01";
   nuevaTarea<="11";

   contadorBucleSalidaMayorNumUF
                           <= conv_std_logic_vector(2**numUFLog,numUFLog+1);

  deboLeerFifoReconfiguracion:
     process( clk, rst, deboLeerReset, deboLeerSet )
        begin
           if (rst='1') then
              deboLeerSalida <= '1';
           elsif (clk'event and clk='1') then
              if (deboLeerReset = '1') then
                deboLeerSalida <= '0';
              elsif (deboLeerSet = '1') then
                deboLeerSalida <= '1';
              else
                deboLeerSalida <= deboLeerSalida;
              end if;
           end if;
     end process;

      stateGen:
      process (escribirFifoReconfiguraciones,
```

```
currentState,fifoEventosFifoVacia, fifoEventosSalidaDatos,
circuitoReconfiguracionLibre, fifoReconfiguracionesFifoVacia,
tablaTareasPreparadaTablaSiguienteOperacion,
codificadorSubtareaACargarAlgunaValeUno, UFestados,
tablaTareasPrepararaParaEjecutar, contadorBucleSalida,
codificadorComprobarDependenciasAlgunaValeUno, tablaTareasAcierto)

begin
  nextState <= currentState;
  case currentState is

        when reposoOLectura =>
           if (fifoEventosFifoVacia='0')  then
                nextState <= distinguirEvento;
           end if;

        when distinguirEvento =>
           if (fifoEventosSalidaDatos(1 downto 0)=nuevaTarea and
                circuitoReconfiguracionLibre='1' and
                fifoReconfiguracionesFifoVacia='0') then
                nextState <= lecturaFifoReconfiguracionesEventoNuevaTarea;
            elsif (fifoEventosSalidaDatos(1 downto 0)=finReconfiguracion
                and tablaTareasPreparadaTablaSiguienteOperacion='0') then
                nextState <= distinguirEvento;
            elsif (fifoEventosSalidaDatos(1 downto 0)=finReconfiguracion
                 and tablaTareasPreparadaTablaSiguienteOperacion='1') then
                nextState <= comprobarDependenciasEventoFinReconfiguracion;
            elsif (fifoEventosSalidaDatos(1 downto 0)=finEjecucion and
                tablaTareasPreparadaTablaSiguienteOperacion='0') then
                nextState <= distinguirEvento;
            elsif (fifoEventosSalidaDatos(1 downto 0)=finEjecucion and
                tablaTareasPreparadaTablaSiguienteOperacion='1') then
                nextState <= actualizarDependencias;
            else
                nextState <= reposoOLectura;
            end if;


        --PRINCIPIO EVENTO NUEVA TAREA

        when lecturaFifoReconfiguracionesEventoNuevaTarea =>
               if (escribirFifoReconfiguraciones='0') then
                   nextState <= distinguirEstadoUnidadEventoNuevaTarea;
               end if;

        when distinguirEstadoUnidadEventoNuevaTarea =>
               if (codificadorSubtareaACargarAlgunaValeUno='1' and
                    UFestados(conv_integer(codificadorSubtareaACargarSalida)*3+
                    3-1 downto conv_integer
                    (codificadorSubtareaACargarSalida)*3)="001" and
                    UFpedir_cargas(conv_integer
                    (codificadorSubtareaACargarSalida))='1') then
                       nextState <= envioComenzarCargaEventoNuevaTarea;
               else
                       nextState <= reposoOLectura;
               end if;

        when envioComenzarCargaEventoNuevaTarea =>
                nextState <= reposoOLectura;


        --FIN EVENTO NUEVA TAREA
        --PRINCIPIO EVENTO FIN RECONFIGURACION

        when comprobarDependenciasEventoFinReconfiguracion =>
                       nextState <= subtareaPreparadaEventoFinReconfiguracion;

        when subtareaPreparadaEventoFinReconfiguracion =>
```

```
      if (codificadorComprobarDependenciasAlgunaValeUno='1'
          and tablaTareasAcierto='1'
          and tablaTareasPrepararaParaEjecutar='1') then
            nextState <= envioComenzarEjecucionEventoFinReconfiguracion;
      elsif (circuitoReconfiguracionLibre='1') then
            nextState <= lecturaFifoReconfiguracionesEventoNuevaTarea;
      else
            nextState <= reposoOLectura;
      end if;

   when envioComenzarEjecucionEventoFinReconfiguracion =>
      if (circuitoReconfiguracionLibre='1') then
         nextState <= lecturaFifoReconfiguracionesEventoNuevaTarea;
      else
         nextState <= reposoOLectura;
      end if;

   --FIN EVENTO FIN RECONFIGURACION
   --PRINCIPIO EVENTO FIN EJECUCION

   when actualizarDependencias =>
      if (circuitoReconfiguracionLibre='1') then
         nextState <= lecturaFifoReconfiguracionesEventoFinEjecucion;
      else
         nextState <= bucleFor;
      end if;


   when lecturaFifoReconfiguracionesEventoFinEjecucion =>
     if (escribirFifoReconfiguraciones='0') then
          nextState <= distinguirEstadoUnidadEventoFinEjecucion;
     end if;

   when distinguirEstadoUnidadEventoFinEjecucion =>
     if (codificadorSubtareaACargarAlgunaValeUno='1' and
        UFestados(conv_integer (codificadorSubtareaACargarSalida)*3+3-1
        downto conv_integer (codificadorSubtareaACargarSalida)*3)="001"
        and UFpedir_cargas
        (conv_integer (codificadorSubtareaACargarSalida))='1') then
           nextState <= envioComenzarCargaEventoFinEjecucion;
     else
          nextState <= bucleFor;
     end if;

   when envioComenzarCargaEventoFinEjecucion =>
     nextState <= bucleFor;

   when bucleFor =>
     if (contadorBucleSalida >= contadorBucleSalidaMayorNumUF) then
          nextState <= reposoOLectura;
     else
          nextState <= comprobarDependenciasEventoFinEjecucion;
     end if;

   when comprobarDependenciasEventoFinEjecucion =>
       if (tablaTareasPreparadaTablaSiguienteOperacion='1') then
            nextState <= subtareaPreparadaEventoFinEjecucion;
       end if;

   when subtareaPreparadaEventoFinEjecucion =>
      if (codificadorComprobarDependenciasAlgunaValeUno='1' and
        tablaTareasAcierto='1' and tablaTareasPrepararaParaEjecutar='1' and
        UFpedir_ejecuciones(conv_integer (contadorBucleSalida(numUFLog-1
        downto 0)))='1') then
          nextState <= envioComenzarEjecucionEventoFinEjecucion;

      else
          nextState <= bucleFor;
```

```
        end if;

      when envioComenzarEjecucionEventoFinEjecucion =>
        nextState <= bucleFor;

            --FIN EVENTO FIN EJECUCION

  end case;
  end process stateGen;

  state:
  process (rst, clk)
  begin
    if (rst = '1') then
      currentState <= reposoOLectura;
    elsif (clk'event and clk='1') then
      currentState <= nextState;
    end if;
  end process state;


  mooreGen:
  process (escribirFifoReconfiguraciones, currentState, fifoEventosFifoVacia,
     deboLeerSalida, fifoReconfiguracionesFifoVacia,
     codificadorComprobarDependenciasSalida,
     contadorBucleSalida(numUFLog-1 downto 0))
   begin
      case currentState is

        when reposoOLectura =>
          fifoEventosLeer <= NOT fifoEventosFifoVacia;
          fifoReconfiguracionesLeer <= '0';
          subtareaACargarComenzarCarga <= '0';
          comprobarDependenciasComenzarEjecucion <= '0';
          tablaTareasBorrar <= '0';
          rstContadorBucle <= '0';
          contadorBucleIncremento <= '0';
          bucle <= '0';
          deboLeerReset <= '0';
          deboLeerSet <= '0';
          decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

      when distinguirEvento =>
          fifoEventosLeer <= '0';
          fifoReconfiguracionesLeer <= '0';
          subtareaACargarComenzarCarga <= '0';
          comprobarDependenciasComenzarEjecucion <= '0';
          tablaTareasBorrar <= '0';
          rstContadorBucle <= '0';
          contadorBucleIncremento <= '0';
          bucle <= '0';
          deboLeerReset <= '0';
          deboLeerSet <= '0';
          decodificadorComprobarDependenciasEntrada <=
                                     codificadorComprobarDependenciasSalida;

      when lecturaFifoReconfiguracionesEventoNuevaTarea =>
         fifoEventosLeer <= '0';
         fifoReconfiguracionesLeer <= deboLeerSalida AND NOT
                                           escribirFifoReconfiguraciones;
         subtareaACargarComenzarCarga <= '0';
         comprobarDependenciasComenzarEjecucion <= '0';
         tablaTareasBorrar <= '0';
         rstContadorBucle <= '0';
         contadorBucleIncremento <= '0';
         bucle <= '0';
         deboLeerReset <= '0';
```

```
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when distinguirEstadoUnidadEventoNuevaTarea =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '1';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when envioComenzarCargaEventoNuevaTarea =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '1';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '1';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when comprobarDependenciasEventoFinReconfiguracion =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when subtareaPreparadaEventoFinReconfiguracion =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when envioComenzarEjecucionEventoFinReconfiguracion =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '1';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
```

```
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when actualizarDependencias =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '1';
            rstContadorBucle <= '1';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                    codificadorComprobarDependenciasSalida;

        when lecturaFifoReconfiguracionesEventoFinEjecucion =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= deboLeerSalida and NOT
                                              escribirFifoReconfiguraciones;
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                        codificadorComprobarDependenciasSalida;

        when distinguirEstadoUnidadEventoFinEjecucion =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '1';
            deboLeerSet <= '0';
            decodificadorComprobarDependenciasEntrada <=
                                        codificadorComprobarDependenciasSalida;

        when envioComenzarCargaEventoFinEjecucion =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '1';
            comprobarDependenciasComenzarEjecucion <= '0';
            tablaTareasBorrar <= '0';
            rstContadorBucle <= '0';
            contadorBucleIncremento <= '0';
            bucle <= '0';
            deboLeerReset <= '0';
            deboLeerSet <= '1';
            decodificadorComprobarDependenciasEntrada <=
                                        codificadorComprobarDependenciasSalida;

        when bucleFor =>
            fifoEventosLeer <= '0';
            fifoReconfiguracionesLeer <= '0';
            subtareaACargarComenzarCarga <= '0';
            comprobarDependenciasComenzarEjecucion <= '0';
```

```
         tablaTareasBorrar <= '0';
         rstContadorBucle <= '0';
         contadorBucleIncremento <= '0';
         bucle <= '1';
         deboLeerReset <= '0';
         deboLeerSet <= '0';
         decodificadorComprobarDependenciasEntrada <=
                              contadorBucleSalida(numUFLog-1 downto 0);

      when comprobarDependenciasEventoFinEjecucion =>
         fifoEventosLeer <= '0';
         fifoReconfiguracionesLeer <= '0';
         subtareaACargarComenzarCarga <= '0';
         comprobarDependenciasComenzarEjecucion <= '0';
         tablaTareasBorrar <= '0';
         rstContadorBucle <= '0';
         contadorBucleIncremento <= '0';
         bucle <= '1';
         deboLeerReset <= '0';
         deboLeerSet <= '0';
         decodificadorComprobarDependenciasEntrada <=
                              contadorBucleSalida(numUFLog-1 downto 0);

      when subtareaPreparadaEventoFinEjecucion =>
         fifoEventosLeer <= '0';
         fifoReconfiguracionesLeer <= '0';
         subtareaACargarComenzarCarga <= '0';
         comprobarDependenciasComenzarEjecucion <= '0';
         tablaTareasBorrar <= '0';
         rstContadorBucle <= '0';
         contadorBucleIncremento <= '1';
         bucle <= '1';
         deboLeerReset <= '0';
         deboLeerSet <= '0';
         decodificadorComprobarDependenciasEntrada <=
                              contadorBucleSalida(numUFLog-1 downto 0);

      when envioComenzarEjecucionEventoFinEjecucion =>
         fifoEventosLeer <= '0';
         fifoReconfiguracionesLeer <= '0';
         subtareaACargarComenzarCarga <= '0';
         comprobarDependenciasComenzarEjecucion <= '1';
         tablaTareasBorrar <= '0';
         rstContadorBucle <= '0';
         contadorBucleIncremento <= '0';
         bucle <= '1';
         deboLeerReset <= '0';
         deboLeerSet <= '0';
         decodificadorComprobarDependenciasEntrada <=
                                          contadorBucleSalidaMenosUno;

     end case;
   end process mooreGen;

   contadorBucleSalidaMenosUno <= contadorBucleSalida(numUFLog-1 downto 0)-1;

     cerosNumBitsEtiqueta <= (others => '0');

 --a la fifo de eventos prioridad en la lectura al resto en la escritura
    arbitro:
    process (fifoEventosLeer, fifoEventosFifoLlena, eventoNuevaTarea,
             UFeventos, UFescribirEnFifoEventos)
      begin
        fifoEventosEscribir <= '0';
        fifoEventosEntradaDatos <= (others => '0');
        UFconcedidaEscritura <= (others => '0');
        concedidaEscrituraNuevaTarea <= '0';
        if (fifoEventosLeer='0') then
```

```
        if (fifoEventosFifoLlena='1') then
            fifoEventosEscribir <= '0';
            fifoEventosEntradaDatos <= (others => '0');
            UFconcedidaEscritura <= (others => '0');
             concedidaEscrituraNuevaTarea <= '0';
        else
           if (eventoNuevaTarea='1') then
               fifoEventosEscribir <= '1';
               fifoEventosEntradaDatos <= cerosNumBitsEtiqueta & nuevaTarea;
               UFconcedidaEscritura <= (others => '0');
                concedidaEscrituraNuevaTarea <= '1';
           else
            --da prioridad a la mas significativa (codificador prioridad)
            for I in UFescribirEnFifoEventos'reverse_range loop
               if UFescribirEnFifoEventos(I)='1' then
                   fifoEventosEscribir <= '1';
            fifoEventosEntradaDatos <=
                    UFeventos(I*(numBitsEtiqueta+2)+(numBitsEtiqueta+2)-1
                                            downto I*(numBitsEtiqueta+2));
            UFconcedidaEscritura <= conv_std_logic_vector(2**I, 2**numUFLog);
            concedidaEscrituraNuevaTarea <= '0';
                end if;
            end loop;
            end if;
           end if;
        end if;

     end process arbitro;

end tablaUFControlFifoEventosArch;
```

FUENTES EN C DE LA VERSIÓN SW

Tabla software

```
#include <stdio.h>
#include <stdlib.h>
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xtmrctr.h"
#include "xbasic_types.h"

#define true 1;
#define false 0;

typedef int bool;

XTmrCtr counter;
XStatus Status;
Xuint32 t1,t2,t3,t4,t5,t6,t7,t8;

// Node Structure
struct table_node
{
	//graph file struct
	int 		node_id;	//nodeid
	double		t_ex;		//time execution
	double		t_res;		//time to reshape
	//predecessors
	int	n_predecessors;
	int	predecessors[9];	//predecessors list
	//next in list
};

struct dep_table
{
	int n_tasks;
	struct table_node task[9];
};

struct dep_table table;

//---------------------------------------------------------------------------

void table_list_init()
{
	table.n_tasks=0;
}
//---------------------------------------------------------------------------
```

```
void table_list_add(int node_id, double t_ex, double t_res, int
predecessors[9], int n_predecessors)
{
	int pos=table.n_tasks;
	//node info
	table.task[pos].node_id=node_id;
	table.task[pos].t_ex=t_ex;
	table.task[pos].t_res=t_res;
	//predecessors
	int i=0;
	for (i=0;i<n_predecessors;i++)
	{
		table.task[pos].predecessors[i]=predecessors[i];
	}
	table.task[pos].n_predecessors=n_predecessors;
	//update n_task
	table.n_tasks=table.n_tasks+1;
}

//---------------------------------------------------------------------------

void table_list_updatedependecies(int task_id)
{
	int i=0;
	int n=table.n_tasks;
	//search predecessors list to delete dependencies
	for (i=0; i<n ;i++)
	{
		int p=table.task[i].n_predecessors;
		int b=-1;
		int j=0;
		//search position
		for (j=0; (j<p && b==-1) ;j++)
		{
			if (table.task[i].predecessors[j]==task_id)
			{
				b=j;
			}
		}
		//update list
		if (b!=-1)
		{
			//backup
			int s=0;
			int backup[9];
			for (j=0; j<p ;j++)
			{
				if (j!=b)
				{
					backup[s]=table.task[i].predecessors[j];
					s=s+1;
				}
			}
			//restore
			table.task[i].n_predecessors=table.task[i].n_predecessors-1;
			for (j=0; j<p ;j++)
			{
				table.task[i].predecessors[j]=backup[j];
			}
		}
	} //for
}
```

```
//---------------------------------------------------------------------------

void table_list_delete(int node_id)
{
	int i=0;
	int n=table.n_tasks;
	int b=-1;
	//search position
	for (i=0; (i<n && b==-1) ;i++)
	{
		if (table.task[i].node_id==node_id)
		{
			b=i;
		}
	}
	if (b!=-1)
	{
		//backup
		int s=0;
		struct table_node backup[9];
		for (i=0; i<n ;i++)
		{
			if (i!=b)
			{
				backup[s]=table.task[i];
				s=s+1;
			}
		}
		//restore
		table.n_tasks=table.n_tasks-1;
		for (i=0; i<n ;i++)
		{
			table.task[i]=backup[i];
		}
	}
	//delete dependencies
	table_list_updatedependecies( node_id);
}

//---------------------------------------------------------------------------

int table_list_task_ready()
{
	int i;
	i=0;
	int n;
	n=table.n_tasks;
	int b;
	b=-1;
	while (i<n&&b==-1)
	{
		int p=table.task[i].n_predecessors;
		if (p==0)
		{
			b=i;
		}
		i++;
	}
	return b;
}
```

```
//---------------------------------------------------------------------------

void table_list_test_load()
{
	table_list_init();

	//task1
	int predecessor[9];
	table_list_add(1,2,2,predecessor,0);

	//task2
	table_list_add(2,2,1,predecessor,0);
	predecessor[0]=1;
	predecessor[1]=2;
	table_list_add(3,3,1,predecessor,2);

	//task4
	table_list_add(4,3,2,predecessor,0);

	//task5
	predecessor[0]=4;
	table_list_add(5,5,2,predecessor,1);

	//task6
	predecessor[0]=3;
	predecessor[1]=5;
	table_list_add(6,3,1,predecessor,2);

	t1=XTmrCtr_GetValue(&counter, 0);
	t2=XTmrCtr_GetValue(&counter, 0);
	xil_printf("Tiempo en cargar el grafo = %d\r\n", t2-t1);
}

//---------------------------------------------------------------------------

void table_list_show ()
{
	int n_tasks=table.n_tasks;
	int i=0;
	int j=0;
	//show tasks
	printf("------------\r\n");
	for (i=0;i<n_tasks;i++)
	{
		//print values
		printf("Task id: %d \r\n",table.task[i].node_id);
		printf("Execution time: %f \r\n",table.task[i].t_ex);
		printf("Reshape time: %f \r\n",table.task[i].t_res);

		//predecessors
		printf("Predecessors: ");
		int n_predecessors=table.task[i].n_predecessors;
		for (j==0;j<n_predecessors;j++)
		{
			printf("%d, ",table.task[i].predecessors[j]);
		}
		printf("\r\n");
	}
	printf("------------\r\n");
}

//---------------------------------------------------------------------------

int main()
{
	Status = XTmrCtr_Initialize(&counter, XPAR_OPB_TIMER_1_DEVICE_ID);
	XTmrCtr_Reset(&counter, 0);
	XTmrCtr_Start(&counter, 0);
```

```
	table_list_test_load();
	while (table.n_tasks>0)
	{
		int b=table_list_task_ready();
		int id=table.task[b].node_id;
		table_list_delete(id);
	}
	print("Fin ejecucion\r\n");
	XTmrCtr_Stop(&counter, 0);
	return 0;
}
```